User's Manual

RZ/A2M Group

DRP Library User's Manual

Renesas Electronics
www.renesas.com

Rev.1.03 May. 2019

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

How to Use This Manual

1. Purpose and Target Readers

This manual is intended to provide the user with an understanding of the functions of the DRP library and how to utilize them. It is aimed at users designing application systems making use of the DRP library. In order to use this manual, you will need a basic knowledge of programming languages and microprocessors.

> Particular attention should be paid to the precautionary notes when using the manual. These notes occur within the body of the text, at the end of each section.

> The revision history summarizes the locations of revisions and additions. It does not list all revisions. Refer to the text of the manual for details.

Contents

1. Introduction

1.1 Summary

This manual describes the functions and usage of the DRP library, which run on the dynamically reconfigurable processor (DRP) of RZ/A2M Group Microprocessors.

The DRP can perform various functions according to user's setting. In this document, the function performed by DRP is called "circuit", and the data representing circuit information is called "configuration data". Writing of the circuit to DRP can be performed by loading the configuration data using DRP Driver[*1]. DRP Library is a collection of configuration data with various functions, mainly image processing.

Note 1. For details of DRP Driver, refer to "RZ/A2M Group DRP Driver User's Manual (R01US0355)".

1.2 Functions

The functions of the configuration data contained in the DRP library are listed below.

Table 1.1 DRP Library Functions

Category	Function Name	Outline	Page
Simple ISP	SimpleIsp	Implements simple image signal processor (ISP) functionality using pipeline processing	14
Image filter	BinarizationFixed	Converts the image to a binary image with a fixed threshold (fixed threshold)	21
	BinarizationAdaptive	Converts the image to a binary image with a dynamic threshold matching the surrounding image (adaptive threshold)	22
	BinarizationAdaptiveBit	Converts the image to a binary image with a dynamic threshold matching the surrounding image (adaptive threshold) (bit output)	26
	Dilate	Dilation of white part in the image	28
	Erode	Erosion of white part in the image	30
	GammaCorrection	Corrects the image with gamma value	32
	GaussianBlur	The image smoothing	34
	MedianBlur	Reduces the noise contained in the image	36
	Sobel	Creates the edge of the image using Sobel filter	38
	Prewitt	Creates the edge of the image using Prewitt filter	40
	Laplacian	Creates the edge of the image using Laplacian filter	42
	UnsharpMasking	The image sharpening	44
	HistogramNormalization	Normalizes the histogram of the image	46
	HistogramNormalizationRgb	Normalizes the histogram of the image (RGB)	49
	Opening[*1]	Removes noise from black portions by shrinkage (erosion) followed by expansion (dilation)	53
	Closing[*1]	Removes noise from white portions by expansion (dilation) followed by shrinkage (erosion)	56
Image conversion	Argb2Grayscale	Converts from ARGB to grayscale	59
	Bayer2Grayscale	Converts from RAW data acquired from CMOS to grayscale	60
	Bayer2Rgb	Converts from RAW data acquired from CMOS to RGB	63
	Bayer2RgbColorCorrection	Converts from RAW data acquired from CMOS camera to RGB (With color correction)	69
	Cropping	Crops a part of the image	72
	CroppingRgb	Crops a part of the image (RGB)	74
	ImageRotate	Rotates the image	75
	ResizeBilinearFixed	Resizes the image (bilinear interpolation, scale factor: 2^n)	78
	ResizeBilinearFixedRgb	Resizes the image (bilinear interpolation, scale factor: 2^n) (RGB)	80
	ResizeBilinear	Resizes the image (bilinear interpolation, scale factor: any)	81
	ResizeNearest	Resizes the image (nearest neighbor interpolation, scale factor: any)	83
	Affine	Performs parallel translation and linear transformation on the image	85

Category	Function Name	Outline	Page
Feature detection	CannyCalculate	Detects the edge of the image using the Canny method (performed by continuous processing of 2 functions)	88
	CannyHysterisis		90
	CornerHarris	Detects the corner contained in the image using the method devised by Chris Harris	92
	CircleFitting	Detects circle from the input image	94
	FindContours	Detects contours in the image and calculates its bounding rectangle	98
	MinutiaeExtract	Extracts minutiae points of fingerprint ridge lines used in fingerprint recognition	102
	MinutiaeDelete	Deletes minutiae points of fingerprint ridge lines used in fingerprint recognition	109
	Thinning	Outputs an image on which thinning has been performed	119
Other	ReedSolomon	Performs error correction using Reed-Solomon codes (fixed primitive polynomial)	123
	ReedSolomonGf8	Performs error correction using GF(2^8) Reed-Solomon codes	125
	Histogram	Generates a histogram from the input image	128

Note 1. This function can be executed by a combination of Dilate and Erode.

2. Operation Conditions

The DRP library operates under the conditions listed below.

Table 2.1 Operation Conditions

Item	Description
Microprocessor	RZ/A2M Group Microprocessors*1 • R7S921051VCBG • R7S921052VCBG • R7S921053VCBG

Note 1. The DRP library operates on RZ/A2M Group Microprocessors equipped with a DRP function module. It will not operate on RZ/A2M Group Microprocessors without a DRP function module.

This library was confirmed to operate in the following development environment:

Renesas e^2 studio 7.3.0

The following toolchain is compatible:

GCC ARM Embedded Toolchain 6-2017-q2-update

3. File Structure

Figure 3.1, Figure 3.2 and Figure 3.3 shows the file structure of configuration data and header files in the DRP library.

r_drp_affine	Affine
r_drp_affine.dat	
r_drp_affine.h	
r_drp_argb2grayscale	ARGB2Grayscale
r_drp_argb2grayscale.dat	
r_drp_argb2grayscale.h	
r_drp_bayer2grayscale	Bayer2Grayscale
r_drp_bayer2grayscale.dat	
r_drp_bayer2grayscale.h	
r_drp_bayer2rgb	Bayer2Rgb
r_drp_bayer2rgb.dat	
r_drp_bayer2rgb.h	
r_drp_bayer2rgb_color_correction	Bayer2RgbColorCorrection
r_drp_bayer2rgb_color_correction.dat	
r_drp_bayer2rgb_color_correction.h	
r_drp_binarization_adaptive	BinarizationAdaptive
r_drp_binarization_adaptive.dat	
r_drp_binarization_adaptive.h	
r_drp_binarization_adaptive_bit	BinarizationAdaptiveBit
r_drp_binarization_adaptive_bit.dat	
r_drp_binarization_adaptive_bit.h	
r_drp_binarization_fixed	BinarizationFixed
r_drp_binarization_fixed.dat	
r_drp_binarization_fixed.h	
r_drp_canny_calculate	CannyCalculate
r_drp_canny_calculate.dat	
r_drp_canny_calculate.h	
r_drp_canny_hysterisis	CannyHysterisis
r_drp_canny_hysterisis.dat	
r_drp_canny_hysterisis.h	
r_drp_circle_fitting	CircleFitting
r_drp_circle_fitting.dat	
r_drp_circle_fitting.h	
r_drp_corner_harris	CornerHarris
r_drp_corner_harris.dat	
r_drp_corner_harris.h	
r_drp_cropping	Cropping
r_drp_cropping.dat	
r_drp_cropping.h	
r_drp_cropping_rgb	CroppingRgb
r_drp_cropping_rgb.dat	
r_drp_cropping_rgb.h	
r_drp_dilate	Dilate
r_drp_dilate.dat	
r_drp_dilate.h	
r_drp_erode	Erode
r_drp_erode.dat	
r_drp_erode.h	

Figure 3.1 File Structure (1/3)

r_drp_find_contours	FindContours
r_drp_find_contours.dat	
r_drp_find_contours.h	
r_drp_gamma_correction	GammaCorrection
r_drp_gamma_correction.dat	
r_drp_gamma_correction.h	
r_drp_gaussian_blur	GaussianBlur
r_drp_gaussian_blur.dat	
r_drp_gaussian_blur.h	
r_drp_histogram	Histogram
r_drp_histogram.dat	
r_drp_histogram.h	
r_drp_histogram_normalization	HistogramNormalization
r_drp_histogram_normalization.dat	
r_drp_histogram_normalization.h	
r_drp_histogram_normalization_rgb	HistogramNormalizationRgb
r_drp_histogram_normalization_rgb.dat	
r_drp_histogram_normalization_rgb.h	
r_drp_image_rotate	ImageRotate
r_drp_image_rotate.dat	
r_drp_image_rotate.h	
r_drp_laplacian	LaplacianFilter
r_drp_laplacian.dat	
r_drp_laplacian.h	
r_drp_median_blur	MedianBlur
r_drp_median_blur.dat	
r_drp_median_blur.h	
r_drp_minutiae_delete	MinutiaeDelete
r_drp_minutiae_delete.dat	
r_drp_minutiae_delete.h	
r_drp_minutiae_extract	MinutiaeExtract
r_drp_minutiae_extract.dat	
r_drp_minutiae_extract.h	
r_drp_prewitt	Prewitt
r_drp_prewitt.dat	
r_drp_prewitt.h	
r_drp_reed_solomon	ReedSolomon
r_drp_reed_solomon.dat	
r_drp_reed_solomon.h	
r_drp_reed_solomon_gf8	ReedSolomonGf8
r_drp_reed_solomon_gf8.dat	
r_drp_reed_solomon_gf8.h	
r_drp_resize_bilinear	ResizeBilinear
r_drp_resize_bilinear.dat	
r_drp_resize_bilinear.h	
r_drp_resize_bilinear_fixed	ResizeBilinearFixed
r_drp_resize_bilinear_fixed.dat	
r_drp_resize_bilinear_fixed.h	
r_drp_resize_bilinear_fixed_rgb	ResizeBilinearFixedRgb
r_drp_resize_bilinear_fixed_rgb.dat	
r_drp_resize_bilinear_fixed_rgb.h	

Figure 3.2 File Structure (2/3)

r_drp_resize_nearest	ResizeNearest
r_drp_resize_nearest.dat	
r_drp_resize_nearest.h	
r_drp_simple_isp	SimpleISP
r_drp_simple_isp_bayer2grayscale_3.dat	
r_drp_simple_isp_bayer2grayscale_6.dat	
r_drp_simple_isp_bayer2yuv_3.dat	
r_drp_simple_isp_bayer2yuv_6.dat	
r_drp_simple_isp.h	
r_drp_sobel	Sobel
r_drp_sobel.dat	
r_drp_sobel.h	
r_drp_thinning	Thinning
r_drp_thinning.dat	
r_drp_thinning.h	
r_drp_unsharp_masking	UnsharpMasking
r_drp_unsharp_masking.dat	
r_drp_unsharp_masking.h	

Figure 3.3 File Structure (3/3)

4. DRP Library Reference

4.1 How to Read the DRP Library Reference

In this section the specifications of the configuration data contained in the DRP library are presented in the format shown below.

Function name*1

Function outline

Configuration data file	The name of the configuration data file. Use the DRP Driver's R_DK2_Load() function to load the data in the DRP.
Supported version	Lists the version of the configuration data that operates under present specification. Use the DRP Driver's R_DK2_GetInfo() function to get the version.
Configuration data size (byte)	Lists the size of the configuration data. Lists all versions, if there are different versions.
Header file	The name of the header file for using the configuration data. Use #include "header file" to include the file.
Parameter	Lists the parameters required by the circuit. Parameters are passed from the CPU to the DRP by means of the DRP driver's R_DK2_Start() function. Parameters are defined as a structure within the header file. Before running the circuit, set the parameters on the CPU side. The data type defined in stdint.h is used. Also, the area where parameters are stored and the area indicated by parameters representing addresses such as 'src' and 'dst' must be located in physical memory. *2
I/O details	Lists the details of the data specified by the parameters. Unless otherwise indicated, the same address may be specified for the input buffer address and output buffer address.
Number of tiles	The number of tiles used by the circuit. The DRP has 6 tiles. The DRP Driver's R_DK2_Load() function is used to assign circuits to tiles.
Segmented processing	Indicates that the function can be processed in parallel by multiple circuits. In parallel processing, the input image is divided up in the vertical direction and processed accordingly. The segmented processing can be executed by utilizing the 6 tiles of DRP and loading multiple configuration data of 3 tiles or less. For details on loading multiple configuration data of 3 tiles or less into DRP, see the explanation of R_DK2_Load () function in "RZ/A2M Group DRP Driver User's Manual".

Example: A case where the input image is divided into three portions in the vertical direction

Description	Describes the specifications of the configuration data.
Note	Additional notes appear here.

Note 1. The function name of configuration data is a character string that can be obtained from the configuration data by using the DRP Driver's R_DK2_GetInfo() function.

Note 2. If the values of physical memory in the area of parameters and input/output data of the circuit are incorrect because the values are in the Cortex-A9 cache, etc., the circuit does not work properly. It must be necessary to clean the cache before calling the DRP driver's R_DK2_Start() function or to allocate the parameters and input/output data of circuit to a non-cached area.

For information on using the API functions of the DRP Driver, refer to "RZ/A2M Group DRP Driver User's Manual (R01US0355)".

4.2 Simple ISP

4.2.1 Simple ISP overview

Simple ISP is an ISP (Image Signal Processor) most suitable for image recognition, and it performs color component accumulation, color correction, demosaicing, noise reduction, sharpening, and gamma correction on captured data (Bayer array). These functions are performed with pipeline processing and then output. This DRP library has been prepared for each output format, and there are two types, YCbCr output and Grayscale output. AE (automatic exposure control) can be realized by adjusting the gain of the CMOS sensor and the shutter speed on the CPU side by using the color component integrated value obtained from Simple ISP.

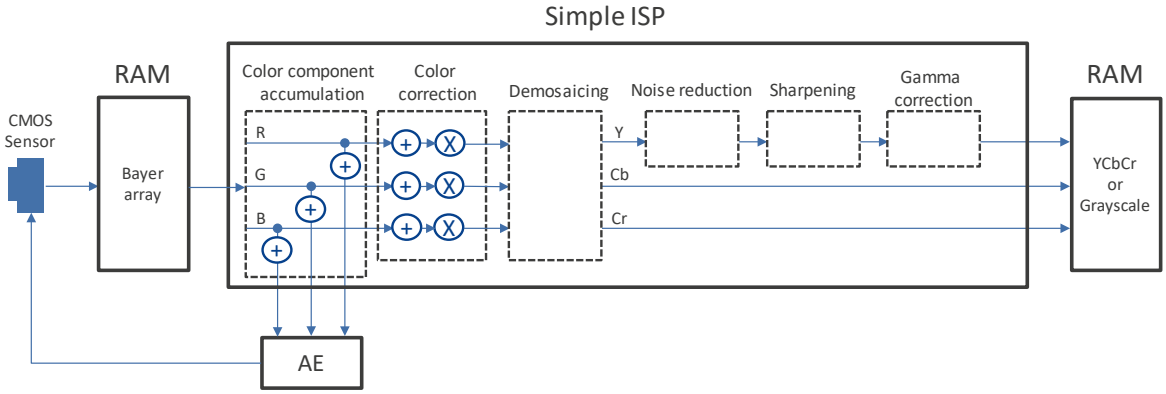


Figure 4.1 Block Diagram of Simple ISP

Color component accumulation	: Accumulated value for each RGB component of Bayer array
Color correction	: Correction by addition and multiplication for each RGB component of Bayer array
Demosaicing	: Interpolation (ACPI / LI) from Bayer array to YCbCr or Y component
Noise reduction	: Noise reduction for Y component (Median filter)
Sharpening	: Sharpening for Y component (Unsharp masking)
Gamma correction	: Gamma correction for Y component

4.2.2 Simple ISP Library structure

The Simple ISP library has configuration data for two types of output format as shown in the table below. Each configuration data file has a 6-tile version optimized for performance and a 3-tile version to suppress the number of tiles, which can be used according to the application.

Table 4.1 Simple ISP Library List

Output format	Tile numbers	Configuration data file name
YCbCr	6 tiles	r_drp_simple_isp_bayer2yuv_6.dat
	3 tiles	r_drp_simple_isp_bayer2yuv_3.dat
Grayscale	6 tiles	r_drp_simple_isp_bayer2grayscale_6.dat
	3 tiles	r_drp_simple_isp_bayer2grayscale_3.dat

4.2.3 Simple Isp API

SimpleIsp

Implements simple image signal processor (ISP) functionality using pipeline processing

Configuration data file			1) r_drp_simple_isp_bayer2yuv_6.dat 2) r_drp_simple_isp_bayer2yuv_3.dat 3) r_drp_simple_isp_bayer2grayscale_6.dat 4) r_drp_simple_isp_bayer2grayscale_3.dat
Supported version			0.91
Configuration data size (byte)			1) 427424, 2) 235328, 3) 369088, 4) 201824
Header file			r_drp_simple_isp.h
Parameter	Structure name		
	r_drp_simple_isp_t		
	Member name	Type	Description
	src	uint32_t	Input image address
	dst	uint32_t	Output image address
	width	uint16_t	Image width (1), 3), and 4) are 16 to 1920, 2) is 16 to 1022, integer multiple of 2)
	height	uint16_t	Image height (4 to 1080, integer multiple of 2)
	component	uint8_t	1: Acquire color component accumulation 0: Do not acquire luminance accumulation
	accumulate	uint32_t	The address of area storing the color component accumulation
	area1_offset_x	uint16_t	x coordinate of the start position of the area 1 for color component accumulation
	area1_offset_y	uint16_t	y coordinate of the start position of the area 1 for color component accumulation
	area1_width	uint16_t	The area 1 for color component accumulation width
	area1_height	uint16_t	The area 1 for color component accumulation height
	area2_offset_x	uint16_t	x coordinate of the start position of the area 2 for color component accumulation
	area2_offset_y	uint16_t	y coordinate of the start position of the area 2 for color component accumulation
	area2_width	uint16_t	The area 2 for color component accumulation width
	area2_height	uint16_t	The area 2 for color component accumulation height
	area3_offset_x	uint16_t	x coordinate of the start position of the area 3 for color component accumulation
	area3_offset_y	uint16_t	y coordinate of the start position of the area 3 for color component accumulation
	area3_width	uint16_t	The area 3 for color component accumulation width
	area3_height	uint16_t	The area 3 for color component accumulation height
	bias_r	int8_t	Bias correction value of image (R component) (-128 to 127)
	bias_g	int8_t	Bias correction value of image (G component) (-128 to 127)
	bias_b	int8_t	Bias correction value of image (B component) (-128 to 127)
	gain_r	uint16_t	Gain correction value of image (R component). The upper 4 bits are an integer part, the lower 12 bits are a decimal part.
	gain_g	uint16_t	Gain correction value of image (G component). The upper 4 bits are an integer part, the lower 12 bits are a decimal part.

	gain_b	uint16_t	Gain correction value of image (B component). The upper 4 bits are an integer part, the lower 12 bits are a decimal part.
	blend	uint16_t	Strength of noise reduction (0x000 to 0x100) 0x000: OFF, 0x100: ON (Maximum)
	strength	uint8_t	Sharpening filter emphasis value (0 to 255)
	coring	uint8_t	Sharpening filter coring value (0 to 255)
	gamma	uint8_t	1: Perform gamma correction 0: Do not perform gamma correction
	table	uint32_t	LUT for gamma correction address
Number of tiles	1) 6, 2) 3, 3) 6, 4) 3		
Segmented processing	Not supported		

Description

Input image

Bayer array of the input image is shown below. The data length of 1 pixel should be 8 bits.

Output image

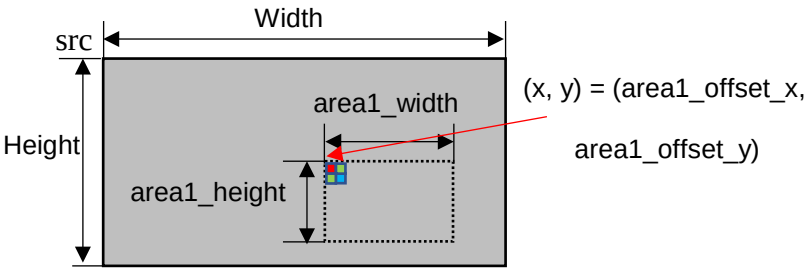
It is output with the image size specified by user settings for YCbCr422 (16 BPP) or Grayscale (8 BPP), parameter "width", "height".

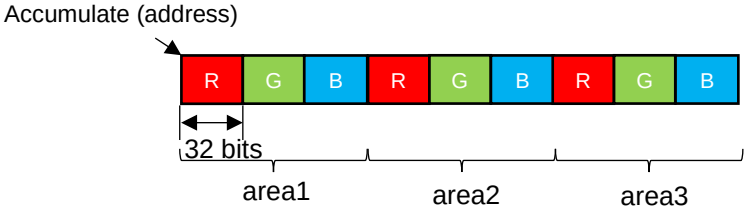
Each pipeline processing details

Color Component Accumulation

It outputs the integrated result for each RGB component of Bayer array. For each of the three areas specified by the parameters area 1 to area 3, it accumulates each of the three components R, G, and B.

A total of nine accumulated values are output to the address specified by the "accumulate" parameter. 1 accumulated value = 32 bit length, please secure a total area of 36 bytes.





From the color component accumulated value, the average luminance can be calculated by the following formula.

$$\text{Average luminance} = \frac{0.299 \times \text{accumulation of R} + 0.587 \times \text{accumulation of G} + 0.114 \times \text{accumulation of B}}{\text{area width} \times \text{area height}}$$

In addition, the average of color components can be calculated by the following formula.

$$\text{Average of color component (R or B)} = \frac{\text{accumulation of R or B}}{\text{area width} \times \text{area height} \div 4}$$

$$\text{Average of color component (G)} = \frac{\text{accumulation of G}}{\text{area width} \times \text{area height} \div 2}$$

Color Correction

For each of the RGB components of the Bayer array, the values set by parameters "bias_r", "bias_g", "bias_b" are added, and the result is then multiplied by the value set by "gain_r", "gain_g", "gain_b".

Demosaicing

For YCbCr output, converts from Bayer array to YCbCr422 by Adaptive Color Plane Interpolation method (ACPI). For Grayscale output, it converts from Bayer array to Grayscale by Linear Interpolation method (LI).

Noise Reduction

Noise reduction is performed by the Median filter algorithm.

You can adjust the amount of noise reduction by combining the input image and the Median filter noise reduction image at the blend ratio designated by the parameter "blend". When 0 is specified for "blend", noise reduction is turned off.

$$\text{Output} = \frac{\text{Input image} \times (256 - \text{blend}) + \text{median image} \times \text{blend}}{256}$$

Sharpening

Sharpens the image using the Unsharp masking algorithm. For input, sharpening is performed by subtracting the edge created by the following 8-direction Laplacian filter. Strength of sharpening is specified as "strength", and threshold of amplitude difference without sharpening is designated by "coring".

8-direction Laplacian filter

1	1	1
1	-8	1
1	1	1

Sharpening processing calculation is as follows.

$$\text{Output} = \text{Input} - \left(\frac{\text{strength}}{256} \times \text{A}\right)$$

A: result of applying 8-direction Laplacian filter

We compare by "coring" so as not to execute sharpening processing on a weak edge with a low amplitude difference. It does not filter on the pixel of interest that satisfies the following formula.

$$coring \geq |A|$$

Gamma Correction

Please store the gamma table at the address specified by parameter "table".
It converts the pixel value by referring to the LUT, which is "table" with values from 0 to 255.

Example

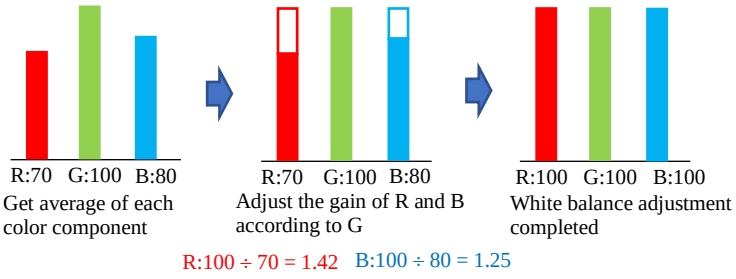
Exposure Control Example

You can calculate the average luminance from the result of color component integration, and perform exposure control using this average luminance. If the average luminance is low, you can adjust this value by decreasing the shutter speed, increasing the gain, increasing the average luminance, increasing the shutter speed, or lowering the gain.

White Balance Example

Using the result of color component accumulation, you can adjust the white balance by performing gain correction as follows.
Based on the G component as a main component, compare the accumulation results of R and B color components and calculate the set value of gain from that ratio.

Example:



In the case of the above example, G is 1.42 times larger than R and G is 1.25 larger than times from B, so set R gain to 1.42 times and B gain to 1.25 times.

Note None

4.3 Image Filter

4.3.1 BinarizationFixed

BinarizationFixed

Converts the image to a binary image with a fixed threshold (fixed threshold)

Configuration data file			r_drp_binarization_fixed.dat	
Supported version			0.90	
Configuration data size (byte)			16960	
Header file			r_drp_binarization_fixed.h	
Parameter	Structure name			
	r_drp_binarization_fixed_t			
	Member name	Type	Description	
	src	uint32_t	Input image address	
	dst	uint32_t	Output image address	
	width	uint16_t	Image width (pixels)	
	height	uint16_t	Image height (pixels)	
	threshold	uint8_t	Binarization threshold (0 to 255)	
I/O details	Input image	Address:	Specified by src.	
		Width (pixels):	Specified by width. (32 to 1280, integer multiple of 8)	
		Height (pixels):	Specified by height. (1 to 960)	
		Format:	8-bit grayscale (1 byte per pixel)	
		Data size:	(width) × (height) × 1 byte	
	Output image	Address:	Specified by dst.	
		Width (pixels):	Same as input image	
		Height (pixels):	Same as input image	
		Format:	8-bit grayscale (0 or 255) (1 byte per pixel)	
		Data size:	(width) × (height) × 1 byte	
Number of tiles	1			
Segmented processing	Supported			
Description	This function binarizes the image at the address specified by src and outputs the result to the address specified by dst. This function outputs 255 when the input data exceeds the threshold (threshold member) and 0 when the input data is equal to or less than the threshold. The processing performed by this function is equivalent to that of the OpenCV cv2::threshold function with thresholdType set to THRESH_BINARY. Reference URL: https://opencv.org/ This function allows the same address to be specified for both src and dst.			
Note	None			

4.3.2 BinarizationAdaptive

BinarizationAdaptive

Converts the image to a binary image with a dynamic threshold matching the surrounding image (adaptive threshold)

Configuration data file			r_drp_binarization_adaptive.dat	
Supported version			0.90	
Configuration data size (byte)			153568	
Header file			r_drp_binarization_adaptive.h	
Parameter	Structure name			
	r_drp_binarization_adaptive_t			
	Member name	Type	Description	
	src	uint32_t	Input image address	
	dst	uint32_t	Output image address	
	width	uint16_t	Image width (pixels)	
	height	uint16_t	Image height (pixels)	
	work	uint32_t	Work area address	
	range	uint8_t	Effective range during average brightness calculation (0 to 255)	
I/O details	Input image	Address:	Specified by src.	
		Width (pixels):	Specified by width. (64 to 1280, integer multiple of 32)	
		Height (pixels):	Specified by height. (40 to 960, integer multiple of 8)	
		Format:	8-bit grayscale (1 byte per pixel)	
		Data size:	(width) × (height) × 1 byte	
	Output image	Address:	Specified by dst.	
		Width (pixels):	Same as input image	
		Height (pixels):	Same as input image	
		Format:	8-bit grayscale (0 or 255) (1 byte per pixel)	
		Data size:	(width) × (height) × 1 byte	
	Work area	Address:	Specified by work.	
		Data size:	(((width × height) ÷ 64) + 2) bytes	
		Description		
		The area used to store average brightness values. Refer to the explanation below for more on average brightness values.		
Number of tiles	3			
Segmented processing	Not supported			

Description

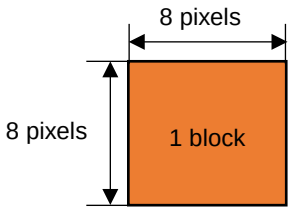
This function binarizes the image at the address specified by src and outputs the result to the address specified by dst.

In the first part of binarization processing, the function divides the input image into blocks of 8 × 8 pixels and calculates the average brightness of each. Then it calculates thresholds from the average brightness values and binarizes the input image.

The method of calculating the average brightness value is as follows. First, blocks of 8 × 8 pixels are delimited, starting from the upper left corner of the input image. Then the maximum and minimum brightness values are sought for each block and the brightness differential is obtained. For blocks where the brightness differential exceeds the range value, the average of the brightness values within the block is used as the average brightness value. For blocks where the brightness differential is equal to or less than the range value, the average brightness value is obtained from the average brightness values of 3 adjacent blocks (above left, above, and left). The method of obtaining the average brightness value is shown in detail below.

(1) Block where the brightness differential exceeds the value of range

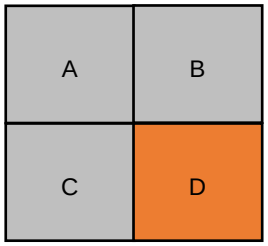
$$\text{Average brightness value} = \text{total brightness values of } 8 \times 8 \text{ pixels} \div 64$$



(2) Block where the brightness differential is equal to or less than the value of range

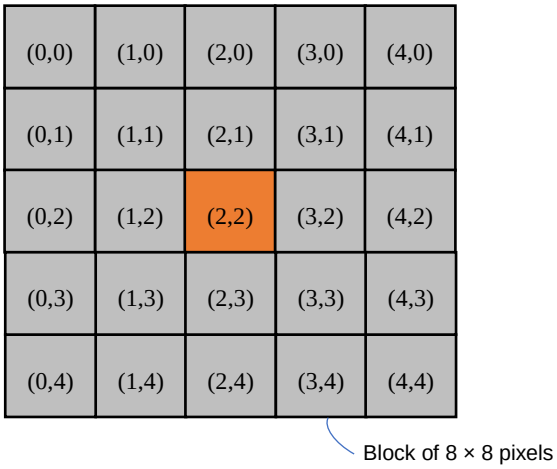
$$\begin{aligned}\text{Average brightness value} &= (\text{average brightness value of A} \\ &+ \text{average brightness value of B} \\ &+ (\text{average brightness value of C} \times 2)) \div 4\end{aligned}$$

However, if the block (D) whose average brightness value we wish to calculate is on the top or left edge of the input image, a value equal to 1/2 the minimum brightness value of D is used because it is not possible to secure average brightness values for the 3 adjacent blocks.



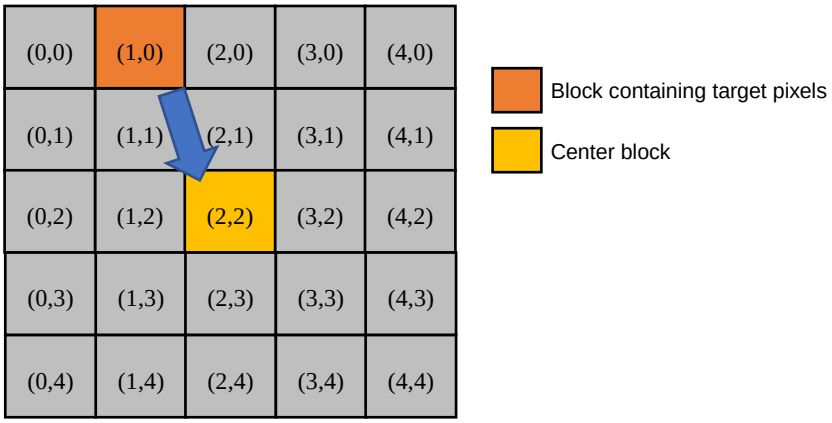
To calculate the thresholds from the average brightness values, groups of 5 × 5 blocks are delimited, each with the block containing the pixels to be binarized (the "target pixels") at the center. The threshold is then calculated from the average brightness values of the group of 5 × 5 blocks. The following equation is used to obtain the threshold.

$$\text{Threshold} = \{(0,0)\ average\ brightness\ value + (1,0)\ average\ brightness\ value + \ldots + (4,4)\ \text{average brightness value}\} \div 25$$



However, if the block containing the target pixels is at the edge of the input image, making it impossible to secure a group of 5 × 5 blocks, the threshold is calculated as described below.

- If the block is within 2 blocks of the top edge
  The block is moved to the center to secure a group of 5 × 5 blocks, and the threshold is calculated.



- If the block is within 2 blocks of the left edge
  0 is used as the threshold.
- If the block is within 2 blocks of the right edge
  The threshold of the block immediately to the left is used.
- If the block is within 2 blocks of the bottom edge
  The threshold of the block immediately above is used.

Note that the results of binarization change as shown below, according to the value specified for range.

Input image Range: 0 Range: 128 Range: 255

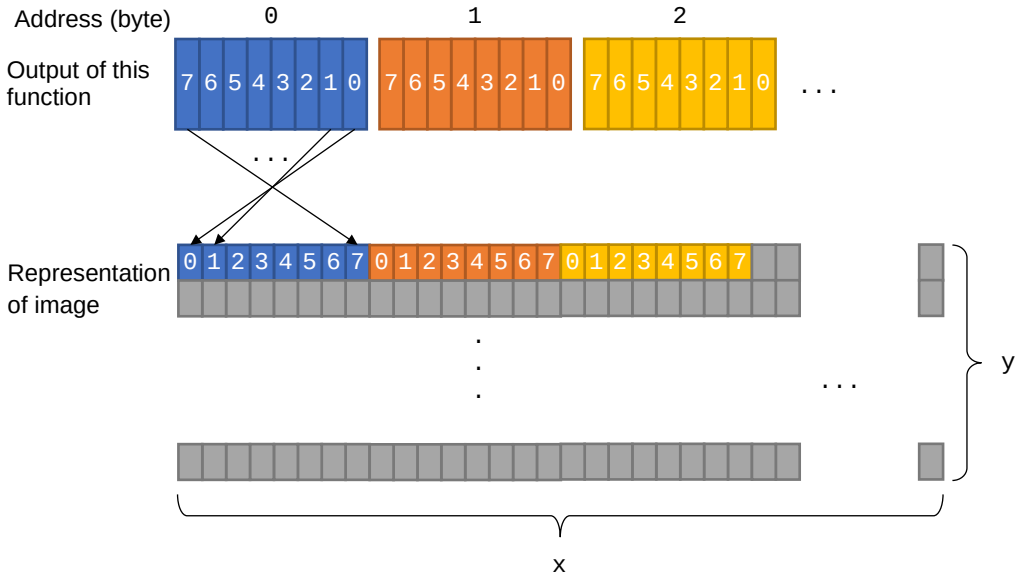
	Using a smaller value for range makes it possible to minimize white blowout and blocked up shadows in the binarized image, but the effects of noise will be more noticeable. Using a larger value for range will reduce the effects of noise but result in more white blowout and blocked up shadows. It is important to set range to a value appropriate for the characteristics of the input image (which are influenced by factors such as the performance of the connected camera and ambient light conditions). This function allows the same address to be specified for both src and dst.
Note	None

4.3.3 BinarizationAdaptiveBit

BinarizationAdaptiveBit

Converts the image to a binary image with a dynamic threshold matching the surrounding image (adaptive threshold) (bit output)

Configuration data file			r_drp_binarization_adaptive_bit.dat	
Supported version			0.90	
Configuration data size (byte)			155968	
Header file			r_drp_binarization_adaptive_bit.h	
Parameter	Structure name			
	r_drp_binarization_adaptive_bit_t			
	Member name	Type	Description	
	src	uint32_t	Input image address	
	dst	uint32_t	Output image address	
	width	uint16_t	Image width (pixels)	
	height	uint16_t	Image height (pixels)	
	work	uint32_t	Work area address	
	range	uint8_t	Effective range during average brightness calculation (0 to 255)	
I/O details	Input image	Address:	Specified by src.	
		Width (pixels):	Specified by width. (64 to 1280, integer multiple of 32)	
		Height (pixels):	Specified by height. (40 to 960, integer multiple of 8)	
		Format:	8-bit grayscale (1 byte per pixel)	
		Data size:	(width) × (height) × 1 byte	
	Output image	Address:	Specified by dst.	
		Width (pixels):	Same as input image	
		Height (pixels):	Same as input image	
		Format:	1 bit per pixel (Refer to the description for details.)	
		Data size:	(width) × (height) ÷ 8 bytes	
	Work area	Address:	Specified by work.	
		Data size:	(((width × height) ÷ 64) + 2) bytes	
		Description		
		The area used to store average brightness values. Refer to the explanation below for more on average brightness values.		
Number of tiles	3			
Segmented processing	Not supported			

Description	This function performs the same processing as that described in 4.2.2, BinarizationAdaptive. It differs from the function described in 4.2.2, BinarizationAdaptive, only in the output format for processing results. The output format of this function uses 1 bit to represent 1 pixel. The arrangement of the bits in the image starts with bit 0 at x coordinate 0, followed by bit 1 at x coordinate 1, and so on. In addition, white is 0 and black is 1.  Setting the range value for this function to 0x18 produces results equivalent to the binarization performed in ZXing ("Zebra Crossing") barcode scanning (implemented by the calculateBlackPoints function and calculateThresholdForBlock function). Reference URL: https://github.com/zxing/zxing This function allows the same address to be specified for both src and dst.
Note	This function differs from the function described in 4.2.2, BinarizationAdaptive, only in the output format for processing results. But when BinarizationAdaptive is a pixel outputting 0, BinarizationAdaptiveBit outputs 1, and when BinarizationAdaptive is a pixel outputting 255, BinarizationAdaptiveBit is 0 Is output. Note this reverse relationship.

4.3.4 Dilate

Dilate

Dilation of white part in the image

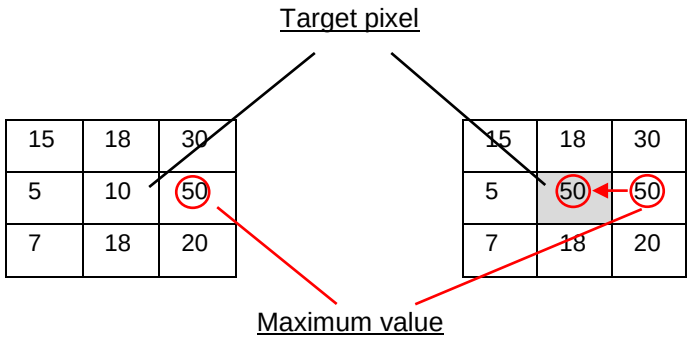
Configuration data file			r_drp_dilate.dat	
Supported version			0.90	
Configuration data size (byte)			56800	
Header file			r_drp_dilate.h	
Parameter	Structure name			
	r_drp_dilate_t			
	Member name	Type	Description	
	src	uint32_t	Input image address	
	dst	uint32_t	Output image address	
	width	uint16_t	Image width (pixels)	
	height	uint16_t	Image height (pixels)	
	top	uint8_t	1: Top edge border processing 0: No top edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the top edge of the source image, otherwise, specify 0.	
	bottom	uint8_t	1: Bottom edge border processing 0: No bottom edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the bottom edge of the source image, otherwise, specify 0.	
I/O details	Input image	Address:	Specified by src.	
		Width (pixels):	Specified by width. (16 to 1280)	
		Height (pixels):	Specified by height. (8 to 960)	
		Format:	8-bit grayscale (1 byte per pixel)	
		Data size:	(width) × (height) × 1 byte	
	Output image	Address:	Specified by dst.	
		Width (pixels):	Same as input image	
		Height (pixels):	Same as input image	
		Format:	8-bit grayscale (1 byte per pixel)	
		Data size:	(width) × (height) × 1 byte	
Number of tiles	1			
Segmented processing	Supported			

Description

This function expands the bright portions of the image at the address specified by src and outputs the result to the address specified by dst.

The maximum value of the 3 × 3 block with the target pixel at the center is set as the new value of the target pixel. When a black and white binary image is input, the white portions appear to expand outward around the pixel. The processing performed is similar to that of the OpenCV cv::dilate() function when border processing is set to BORDER_REPLICATE.

Reference URL: https://opencv.org/



A processing example using a binarized image as the input image is shown below.

Input image Output image

This function allows specification of the same address for both src and dst as long as the processing is not segmented.

Note None

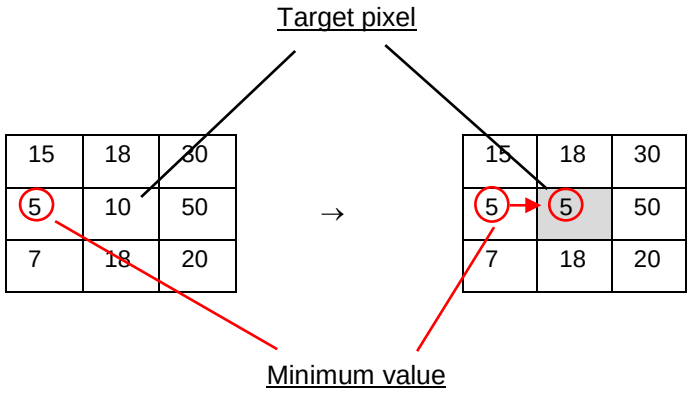
4.3.5 Erode

Erode

Erosion of white part in the image

Configuration data file			r_drp_erode.dat	
Supported version			0.90	
Configuration data size (byte)			60480	
Header file			r_drp_erode.h	
Parameter	Structure name			
	r_drp_erode_t			
	Member name	Type	Description	
	src	uint32_t	Input image address	
	dst	uint32_t	Output image address	
	width	uint16_t	Image width (pixels)	
	height	uint16_t	Image height (pixels)	
	top	uint8_t	1: Top edge border processing 0: No top edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the top edge of the source image, otherwise, specify 0.	
	bottom	uint8_t	1: Bottom edge border processing 0: No bottom edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the bottom edge of the source image, otherwise, specify 0.	
I/O details	Input image	Address:	Specified by src.	
		Width (pixels):	Specified by width. (16 to 1280)	
		Height (pixels):	Specified by height. (8 to 960)	
		Format:	8-bit grayscale (1 byte per pixel)	
		Data size:	(width) × (height) × 1 byte	
	Output image	Address:	Specified by dst.	
		Width (pixels):	Same as input image	
		Height (pixels):	Same as input image	
		Format:	8-bit grayscale (1 byte per pixel)	
		Data size:	(width) × (height) × 1 byte	
Number of tiles	1			
Segmented processing	Supported			

Description	This function contracts the bright portions of the image at the address specified by src and outputs the result to the address specified by dst. The maximum value of the 3 × 3 block with the target pixel at the center is set as the new value of the target pixel. When a black and white binary image is input, the white portions appear to contract outward around the pixel. The processing performed is similar to that of the OpenCV cv::erode() function when border processing is set to BORDER_REPLICATE. Reference URL: https://opencv.org/



A processing example using a binarized image as the input image is shown below.

Input image Output image

This function allows specification of the same address for both src and dst as long as the processing is not segmented.

Note	None

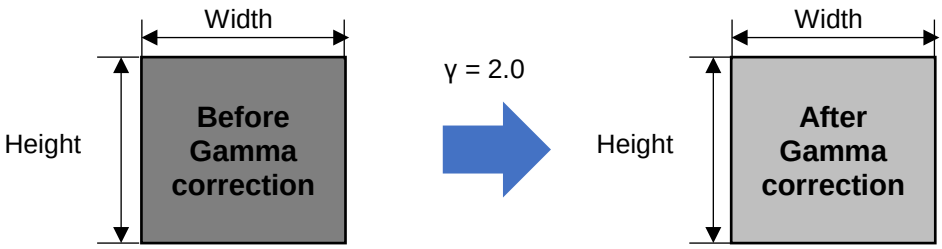
4.3.6 GammaCorrection

GammaCorrection

Corrects the image with gamma value

Configuration data file			r_drp_gamma_correction.dat
Supported version			0.90
Configuration data size (byte)			13120
Header file			r_drp_gamma_correction.h
Parameter	Structure name		
	r_drp_gamma_correction_t		
	Member name	Type	Description
	src	uint32_t	Input image address
	dst	uint32_t	Output image address
	width	uint16_t	Image width (pixels)
	height	uint16_t	Image height (pixels)
I/O details	Input image	Address:	Specified by src.
		Width (pixels):	Specified by width. (16 to 1280, integer multiple of 4)
		Height (pixels):	Specified by height. (1 to 960)
		Format:	8-bit grayscale (1 byte per pixel)
		Data size:	(width) × (height) × 1 byte
	Output image	Address:	Specified by dst.
		Width (pixels):	Same as input image
		Height (pixels):	Same as input image
		Format:	8-bit grayscale (1 byte per pixel)
		Data size:	(width) × (height) × 1 byte
Number of tiles	1		
Segmented processing	Supported		

Description	This function applies Gamma correction to the image at the address specified by src and outputs the result to the address specified by dst.



The function performs Gamma correction by obtaining post-correction brightness values from a lookup table based on a Gamma correction (γ) of 2.0. Post-correction brightness values are calculated using the equation below, where the Gamma correction value is represented as γ, the pre-correction brightness value as src, and the post-correction brightness value as dst.

$$\mathrm{dst} = \left(\frac{src}{255}\right)^{\frac{1}{\gamma}} \times 255$$

For the calculation results using the above equation when γ = 2.0, the value of dst is rounded off after the decimal point. Some examples of src values and their corresponding dst output values are shown below.

src	0	1	2	3	...	253	254	255
dst	0	16	23	28	...	254	254	255

This function allows the same address to be specified for both src and dst.

Note	None

4.3.7 GaussianBlur

GaussianBlur

The image smoothing

Configuration data file			r_drp_gaussian_blur.dat
Supported version			0.90
Configuration data size (byte)			60992
Header file			r_drp_gaussian_blur.h
Parameter	Structure name		
	r_drp_gaussian_blur_t		
	Member name	Type	Description
	src	uint32_t	Input image address
	dst	uint32_t	Output image address
	width	uint16_t	Image width (pixels)
	height	uint16_t	Image height (pixels)
	top	uint8_t	1: Top edge border processing 0: No top edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the top edge of the source image, otherwise, specify 0.
	bottom	uint8_t	1: Bottom edge border processing 0: No bottom edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the bottom edge of the source image, otherwise, specify 0.
I/O details	Input image	Address:	Specified by src.
		Width (pixels):	Specified by width. (16 to 1280)
		Height (pixels):	Specified by height. (8 to 960)
		Format:	8-bit grayscale (1 byte per pixel)
		Data size:	(width) × (height) × 1 byte
	Output image	Address:	Specified by dst.
		Width (pixels):	Same as input image
		Height (pixels):	Same as input image
		Format:	8-bit grayscale (1 byte per pixel)
		Data size:	(width) × (height) × 1 byte

Number of tiles	1
Segmented processing	Supported
Description	This function uses a Gaussian filter to smooth the image at the address specified by src and outputs the result to the address specified by dst. A Gaussian filter is a type of filter used for image smoothing. It uses a Gaussian distribution in which the pixels closest to the target pixel are given the most weight. This function uses the following kernel. 1/16 2/16 1/16 2/16 4/16 2/16 1/16 2/16 1/16 To calculate the value of the target pixel, weighted addition is performed based on a 3 × 3 pixels kernel with the target pixel at the center. The processing performed by this function is equivalent to that of the OpenCV cv::GaussianBlur function with the specification of 3 for ksize.width, 3 for ksize.height, 1.3 for sigmaX, 1.3 for sigmaY, and BORDER_REFLECT_101 for borderType. Reference URL: https://opencv.org/ This function allows specification of the same address for both src and dst as long as the processing is not segmented.
Note	None

4.3.8 MedianBlur

MedianBlur

Reduces the noise contained in the image

Configuration data file			r_drp_median_blur.dat
Supported version			0.90
Configuration data size (byte)			57536
Header file			r_drp_median_blur.h
Parameter	Structure name		
	r_drp_gaussian_blur_t		
	Member name	Type	Description
	src	uint32_t	Input image address
	dst	uint32_t	Output image address
	width	uint16_t	Image width (pixels)
	height	uint16_t	Image height (pixels)
	top	uint8_t	1: Top edge border processing 0: No top edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the top edge of the source image, otherwise, specify 0.
	bottom	uint8_t	1: Bottom edge border processing 0: No bottom edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the bottom edge of the source image, otherwise, specify 0.
I/O details	Input image	Address:	Specified by src.
		Width (pixels):	Specified by width. (24 to 1280)
		Height (pixels):	Specified by height. (8 to 960)
		Format:	8-bit grayscale (1 byte per pixel)
		Data size:	(width) × (height) × 1 byte
	Output image	Address:	Specified by dst.
		Width (pixels):	Same as input image
		Height (pixels):	Same as input image
		Format:	8-bit grayscale (1 byte per pixel)
		Data size:	(width) × (height) × 1 byte
Number of tiles	1		
Segmented processing	Supported		

Description	This function uses a median filter to smooth the image at the address specified by src and outputs the result to the address specified by dst. A median filter is a type of nonlinear digital filter that is widely used to eliminate noise from images or signals. The function replaces the value of the target pixel with the median value of a 9-pixel block with the target pixel at its center. The 9-pixel block consists of a grid of 3 × 3 pixels with the target pixel at its center. The processing performed by this function is equivalent to that of the OpenCV cv::medianBlur function with 3 specified for the argument ksize. Reference URL: https://opencv.org/ This function allows specification of the same address for both src and dst as long as the processing is not segmented.
Note	None

4.3.9 Sobel

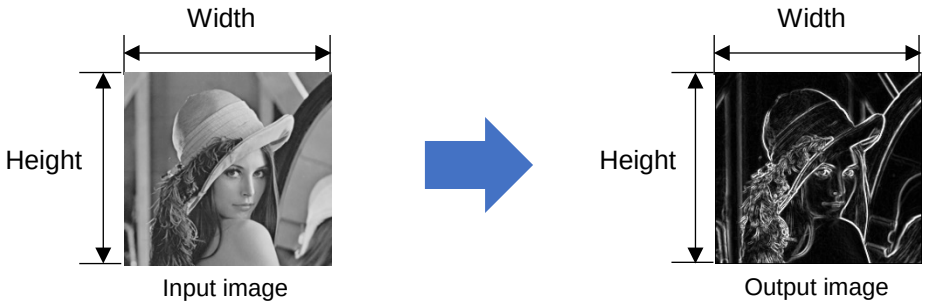
Sobel

Creates the edge of the image using Sobel filter

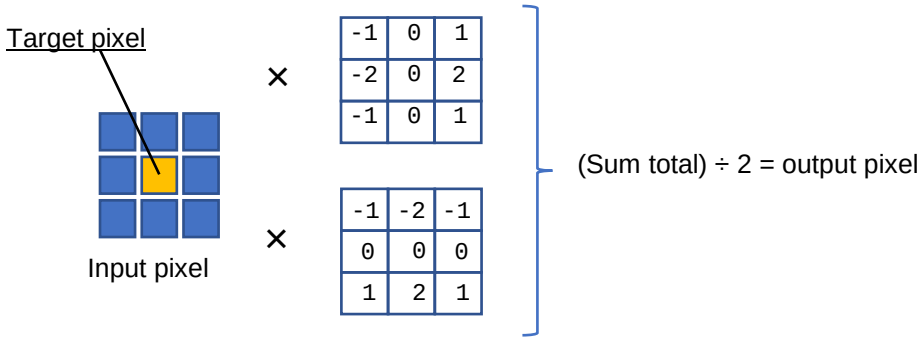
Configuration data file			r_drp_sobel.dat	
Supported version			0.90	
Configuration data size (byte)			40352	
Header file			r_drp_sobel.h	
Parameter	Structure name			
	r_drp_sobel_t			
	Member name	Type	Description	
	src	uint32_t	Input image address	
	dst	uint32_t	Output image address	
	width	uint16_t	Image width (pixels)	
	height	uint16_t	Image height (pixels)	
	top	uint8_t	1: Top edge border processing 0: No top edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the top edge of the source image, otherwise, specify 0.	
	bottom	uint8_t	1: Bottom edge border processing 0: No bottom edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the bottom edge of the source image, otherwise, specify 0.	
I/O details	Input image	Address:	Specified by src.	
		Width (pixels):	Specified by width. (16 to 1280)	
		Height (pixels):	Specified by height. (8 to 960)	
		Format:	8-bit grayscale (1 byte per pixel)	
		Data size:	(width) × (height) × 1 byte	
	Output image	Address:	Specified by dst.	
		Width (pixels):	Same as input image	
		Height (pixels):	Same as input image	
		Format:	8-bit grayscale (1 byte per pixel)	
		Data size:	(width) × (height) × 1 byte	
Number of tiles	1			
Segmented processing	Supported			

Description

This function uses a Sobel filter to emphasize the edges in the image at the address specified by src and outputs the result to the address specified by dst.



The function performs the calculations shown below on a 1 pixel band around the target pixel (an area of 3 × 3 pixels) in order to emphasize edges in the horizontal and vertical directions.



This function allows specification of the same address for both src and dst as long as the processing is not segmented.

Note None

4.3.10 Prewitt

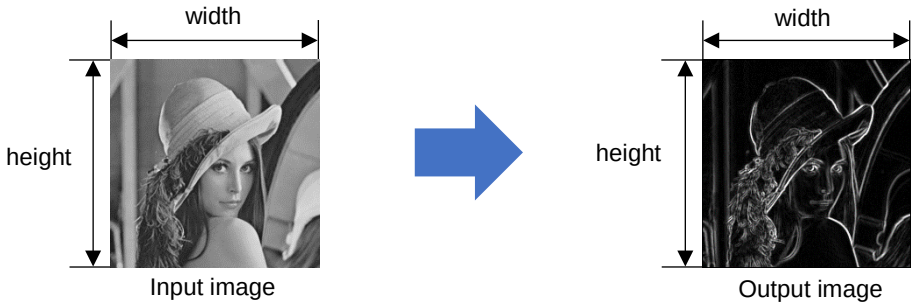
Prewitt

Creates the edge of the image using Prewitt filter

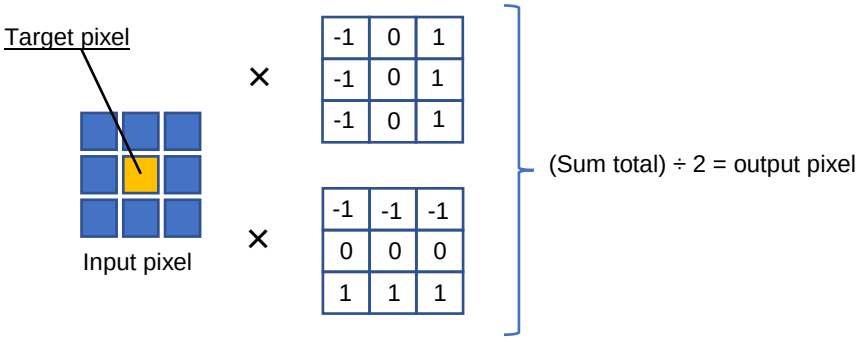
Configuration data file			r_drp_prewitt.dat	
Supported version			0.90	
Configuration data size (byte)			40256	
Header file			r_drp_prewitt.h	
Parameter	Structure name			
	r_drp_prewitt_t			
	Member name	Type	Description	
	src	uint32_t	Input image address	
	dst	uint32_t	Output image address	
	width	uint16_t	Image width (pixels)	
	height	uint16_t	Image height (pixels)	
	top	uint8_t	1: Top edge border processing 0: No top edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the top edge of the source image, otherwise, specify 0.	
	bottom	uint8_t	1: Bottom edge border processing 0: No bottom edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the bottom edge of the source image, otherwise, specify 0.	
I/O details	Input image	Address:	Specified by src.	
		Width (pixels):	Specified by width. (16 to 1280)	
		Height (pixels):	Specified by height. (8 to 960)	
		Format:	8-bit grayscale (1 byte per pixel)	
		Data size:	(width) × (height) × 1 byte	
	Output image	Address:	Specified by dst.	
		Width (pixels):	Same as input image	
		Height (pixels):	Same as input image	
		Format:	8-bit grayscale (1 byte per pixel)	
		Data size:	(width) × (height) × 1 byte	
Number of tiles	1			
Segmented processing	Supported			

Description

This function uses a Prewitt filter to emphasize the edges in the image at the address specified by src and outputs the result to the address specified by dst.



The function performs the calculations shown below on a 1 pixel band around the target pixel (an area of 3 × 3 pixels) in order to emphasize edges in the horizontal and vertical directions.



This function allows specification of the same address for both src and dst as long as the processing is not segmented.

Note None

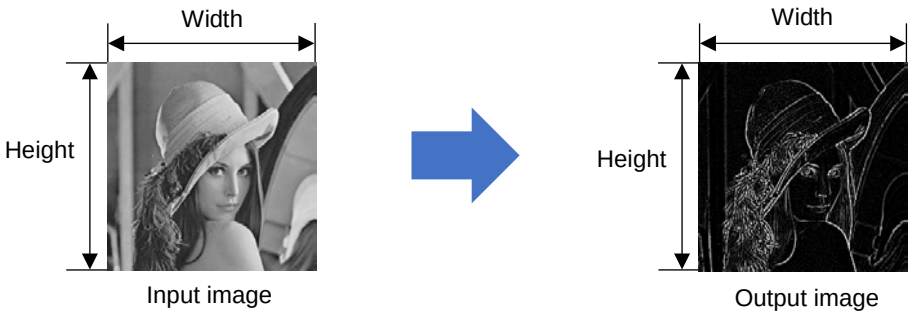
4.3.11 Laplacian

Laplacian

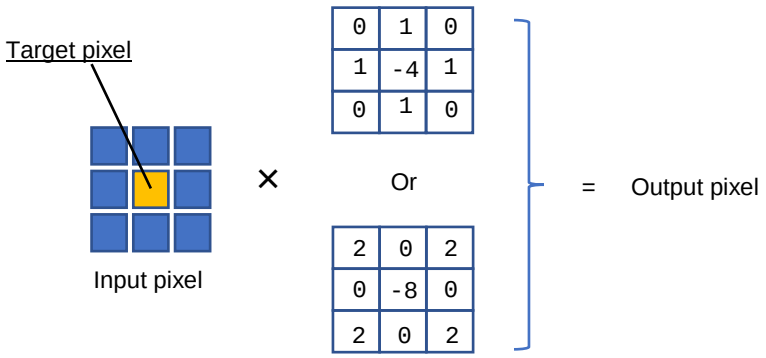
Creates the edge of the image using Laplacian filter

Configuration data file			r_drp_laplacian.dat
Supported version			0.90
Configuration data size (byte)			36544
Header file			r_drp_laplacian.h
Parameter	Structure name		
	r_drp_laplacian_t		
	Member name	Type	Description
	src	uint32_t	Input image address
	dst	uint32_t	Output image address
	width	uint16_t	Image width (pixels)
	height	uint16_t	Image height (pixels)
	top	uint8_t	1: Top edge border processing 0: No top edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the top edge of the source image, otherwise, specify 0.
	bottom	uint8_t	1: Bottom edge border processing 0: No bottom edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the bottom edge of the source image, otherwise, specify 0.
	kernel	uint8_t	0: Use $\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$ as filter coefficients 1: Use $\begin{pmatrix} 2 & 0 & 2 \\ 0 & -8 & 0 \\ 2 & 0 & 2 \end{pmatrix}$ as filter coefficients Specify either 0 or 1.
I/O details	Input image	Address:	Specified by src.
		Width (pixels):	Specified by width. (16 to 1280)
		Height (pixels):	Specified by height. (8 to 960)
		Format:	8-bit grayscale (1 byte per pixel)
		Data size:	(width) × (height) × 1 byte
	Output image	Address:	Specified by dst.
		Width (pixels):	Same as input image
		Height (pixels):	Same as input image
		Format:	8-bit grayscale (1 byte per pixel)
		Data size:	(width) × (height) × 1 byte
Number of tiles	1		
Segmented processing	Supported		

Description	This function uses a Laplacian filter to emphasize the edges in the image at the address specified by src and outputs the result to the address specified by dst.



This function performs the calculations shown below on a 1 pixel band around the target pixel (an area of 3 × 3 pixels) in order to emphasize edges.



The results produced by this function are equivalent to those of the OpenCV cv::Laplacian function with ddepth set to CV_8U, ksize set to 1 or 3, scale set to 1.0, delta set to 0, and borderType set to BORDER_REFLECT_101. The setting ksize = 1 is equivalent to kernel = 0 in this function, and ksize = 3 is equivalent to kernel = 1.

Reference URL: https://opencv.org/

This function allows specification of the same address for both src and dst as long as the processing is not segmented.

Note	None

4.3.12 UnsharpMasking

UnsharpMasking

The image sharpening

Configuration data file			r_drp_unsharp_masking.dat
Supported version			0.90
Configuration data size (byte)			156512
Header file			r_drp_unsharp_masking.h
Parameter	Structure name		
	r_drp_unsharp_masking_t		
	Member name	Type	Description
	src	uint32_t	Input image address
	dst	uint32_t	Output image address
	width	uint16_t	Image width (pixels)
	height	uint16_t	Image height (pixels)
	strength	uint8_t	Filter emphasis value (0 to 255) Refer to the description for details.
	top	uint8_t	1: Top edge border processing 0: No top edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the top edge of the source image, otherwise, specify 0.
	bottom	uint8_t	1: Bottom edge border processing 0: No bottom edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the bottom edge of the source image, otherwise, specify 0.
I/O details	Input image	Address:	Specified by src.
		Width (pixels):	Specified by width. (16 to 1280)
		Height (pixels):	Specified by height. (8 to 960)
		Format:	8-bit grayscale (1 byte per pixel)
		Data size:	(width) × (height) × 1 byte
	Output image	Address:	Specified by dst.
		Width (pixels):	Same as input image
		Height (pixels):	Same as input image
		Format:	8-bit grayscale (1 byte per pixel)
		Data size:	(width) × (height) × 1 byte
Number of tiles	2		
Segmented processing	Supported		

Description	This function sharpens (enhances the edges in) the image at the address specified by src and outputs the result to the address specified by dst. The amount of emphasis can be adjusted using the strength parameter. A larger strength value corresponds to more emphasis of the edges in the image. For UnsharpMasking, the coefficients below used by the OpenCV cv::filter2D() function are typical. (k is the coefficient representing sharpening strength. A value of 0 means no sharpening.)

–k/9	–k/9	–k/9
–k/9	1 + (8 * k/9)	–k/9
–k/9	–k/9	–k/9

Reference URL: https://opencv.org/

This function uses the coefficients below, which approximate the above coefficients using a fixed decimal. k' is specified as strength.

–k'/256	–k'/256	–k'/256
–k'/256	(9 * 28 + (8 * k'))/256	–k'/'256
–k'/256	–k'/256	–k'/256

By specifying a value 28 times the k value as strength, UnsharpMasking can be performed. For example, if a value of 28 is specified for strength, the result would be equivalent to performing UnsharpMasking when k = 1.0.

This function allows specification of the same address for both src and dst as long as the processing is not segmented.

Note	None

4.3.13 HistogramNormalization

HistogramNormalization

Normalizes the histogram of the image

Configuration data file			r_drp_histogram_normalization.dat
Supported version			0.90
Configuration data size (byte)			40128
Header file			r_drp_histogram_normalization.h
Parameter	Structure name		
	r_drp_histogram_normalization_t		
	Member name	Type	Description
	src	uint32_t	Input image address
	dst	uint32_t	Output data address / image address
	width	uint16_t	Image width (pixels)
	height	uint16_t	Image height (pixels)
	mode	uint8_t	1: MODE1 (Survey the overall brightness of the image) 2: MODE2 (Normalize the image) Refer to the description for details
	The followings are the parameters for MODE2 (Please set 0 in MODE1)		
	src_pixel_mean	uint32_t	Mean of the pixel values in input image The upper 20 bits are an integer part, the lower 12 bits are a decimal part. Refer to the description for details
	src_pixel_rstd	uint32_t	Reciprocal of standard deviation of the pixel values in input image The upper 20 bits are an integer part, the lower 12 bits are a decimal part. Refer to the description for details
	dst_pixel_mean	uint8_t	Mean of the pixel values in output image
	dst_pixel_std	uint8_t	Standard deviation of the pixel values in output image
I/O details	Input image	Address: Width (pixels): Height (pixels): Format: Data size:	Specified by src. Specified by width. (8 to 1280, integer multiple of 8) Specified by height. (8 to 960) 8-bit grayscale (1 byte per pixel) (width) × (height) × 1 byte
	Output data (MODE1)	Address: Format: Data size:	Specified by dst. From the top address, specifications are made in the following order. Sum of pixel values (8 bytes) Square-sum of pixel values (8 bytes) 16 bytes
	Output image (MODE2)	Address: Width (pixels): Height (pixels): Format: Data size:	Specified by dst. Same as input image Same as input image 8-bit grayscale (0 or 255) (1 byte per pixel) (width) × (height) × 1 byte
Number of tiles	1		
Segmented processing	Supported Segmented processing can be set up in combination with processing by the CPU. Refer to the description for details.		

Description

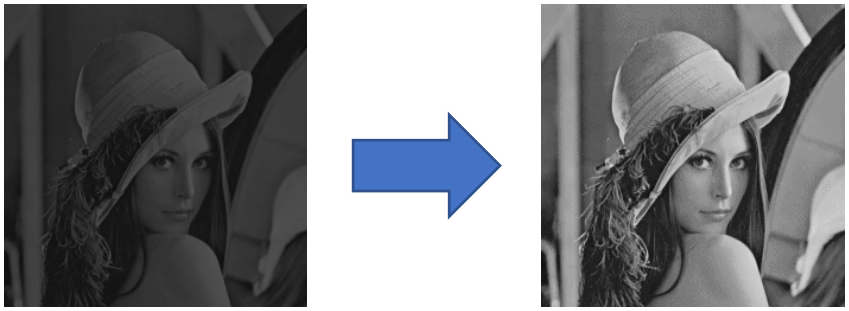
This function has following two operation modes, and when used in combination, it is possible to outputs image normalized the histogram of the input image.

MODE1: Calculates sum and square-sum of the image at the address specified by src and outputs the result to the address specified by dst.

MODE2: Normalizes the image at the address specified by src by following calculation and outputs the result to the address specified by dst.

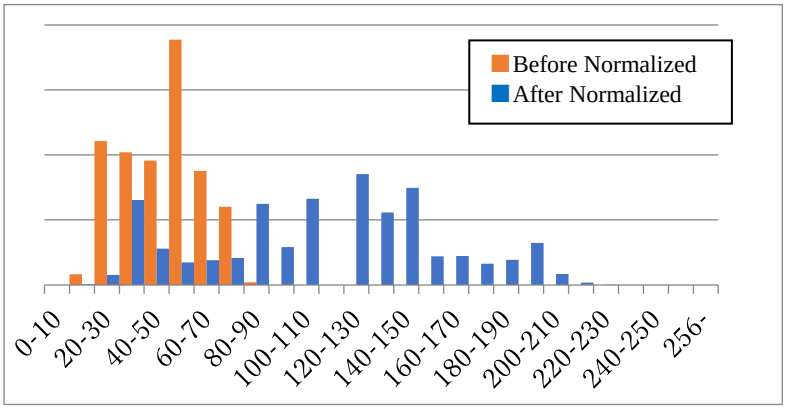
$$\text{output pixel value} = \{(\text{input pixel value} - \text{src\_pixel\_mean}) \times \text{src\_pixel\_rstd}\} \times \text{dst\_pixel\_std} + \text{dst\_pixel\_mean}$$

Please set the target values of the mean and the standard deviation of the normalized image to dst_pixel_mean and dst_pixel_std. When you set dst_pixel_mean = 112 and dst_pixel_std = 48 and normalize the lower left image, this function outputs lower right image.



Input image Output image

Also following figure is the histogram of the input image (before normalized) and the output image (after normalized)



The histogram of the input image and the output image

Follow the steps below to obtain the output image normalized the histogram of the input image.
1. Calculates sum and square-sum of pixel values by executing MODE 1.
2. Calculates src_pixel_mean and src_pixel_rstd using the output result of 1. and the following equations in the CPU.
3. Outputs normalized image by executing MODE2 using the calculation results of 2. as parameters.

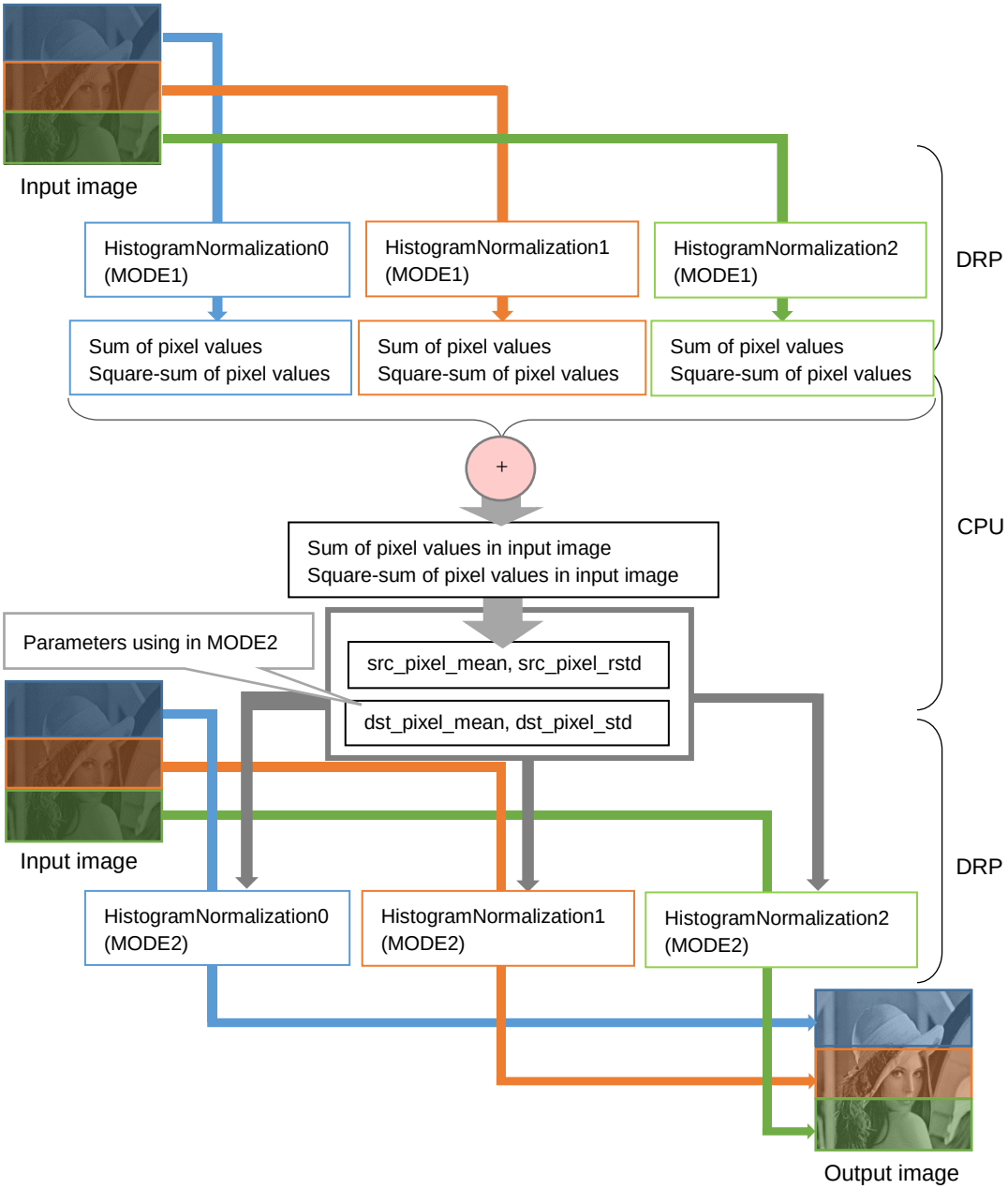
$$\text{src\_pixel\_mean} = \text{sum of pixel value} \div (\text{width} \times \text{height}) \times 4096$$

$$\text{src\_pixel\_rstd} = 1 \div \left(\sqrt{\text{square-sum of pixel value} \div (\text{width} \times \text{height}) - \text{src\_pixel\_mean}^2}\right) \times 4096$$

Because src_pixel_mean and src_pxiel_rstd are fixed point (The upper 20 bits are integer part and lower 12 bits are a decimal part), please be sure to multiply 4096 like the above equations.

This function can execute as segmented processing in combination with CPU processing. An example of three parallel processing flow is shown below.

1. The input data are segmented into three areas. Specify prescribed src, dst, width and height for HistogramNormalization of respective areas. And specify mode 1
2. After DRP processing is complete, calculate src_pixel_mean and src_pixel_rstd using the sum and the square-sum of the pixel values in the dst area of each HistogramNormalization in the CPU.
3. The input data are segmented into three areas. Specify prescribed src, dst, width and height for HistogramNormalization of respective areas. Also, as common parameters, specify the calculation results of 2. for src_pixel_mean and src_pixel_rstd, and arbitrary values for dst_pixel_mean and dst_pixel_std, and 2 for mode.
4. After DRP processing is complete, outputs the normalized image.



This function allows the same address to be specified for both src and dst in MODE2.

Note Do not set values other than those calculated by the equations described in the Description to src_pixel_mean and src_pixel_rstd because it may cause malfunction.

4.3.14 HistogramNormalizationRgb

HistogramNormalizationRgb

Normalizes the histogram of the image (RGB)

Configuration data file			r_drp_histogram_normalization_rgb.dat
Supported version			0.90
Configuration data size (byte)			56224
Header file			r_drp_histogram_normalization_rgb.h
Parameter	Structure name		
	r_drp_histogram_normalization_rgb_t		
	Member name	Type	Description
	src	uint32_t	Input image address
	dst	uint32_t	Output data address / image address
	width	uint16_t	Image width (pixels)
	height	uint16_t	Image height (pixels)
	mode	uint8_t	1: MODE1 (Survey the overall brightness of the image) 2: MODE2 (Normalize the image) Refer to the description for details
	The followings are the parameters for MODE2 (Please set 0 in MODE1)		
	src_pixel_red_mean	uint32_t	Mean of the R component pixel values in input image The upper 20 bits are an integer part, the lower 12 bits are a decimal part. Refer to the description for details
	src_pixel_red_rstd	uint32_t	Reciprocal of standard deviation of the R component pixel values in input image The upper 20 bits are an integer part, the lower 12 bits are a decimal part. Refer to the description for details
	src_pixel_green_mean	uint32_t	Mean of the G component pixel values in input image The upper 20 bits are an integer part, the lower 12 bits are a decimal part. Refer to the description for details
	src_pixel_green rstd	uint32_t	Reciprocal of standard deviation of the G component pixel values in input image The upper 20 bits are an integer part, the lower 12 bits are a decimal part. Refer to the description for details
	src_pixel_blue_mean	uint32_t	Mean of the B component pixel values in input image The upper 20 bits are an integer part, the lower 12 bits are a decimal part. Refer to the description for details
	src_pixel_blue_ rstd	uint32_t	Reciprocal of standard deviation of the B component pixel values in input image The upper 20 bits are an integer part, the lower 12 bits are a decimal part. Refer to the description for details
	dst_output_mean	uint8_t	Mean of the pixel values in output image
	dst_output_std	uint8_t	Standard deviation of the pixel values in output image
I/O details	Input image	Address:	Specified by src.
		Width (pixels):	Specified by width. (8 to 1280, integer multiple of 8)
		Height (pixels):	Specified by height. (8 to 960)
		Format:	RGB (3 bytes per pixel)
		Data size:	(width) × (height) × 3 bytes

	Output data (MODE1)	Address:	Specified by dst.
		Format:	From the top address, specifications are made in the following order. Sum of R component pixel values (8 bytes) Square-sum of R component pixel values (8 bytes) Sum of G component pixel values (8 bytes) Square-sum of G component pixel values (8 bytes) Sum of B component pixel values (8 bytes) Square-sum of B component pixel values (8 bytes)
		Data size:	48 bytes
	Output image (MODE2)	Address:	Specified by dst.
		Width (pixels):	Same as input image
		Height (pixels):	Same as input image
		Format:	RGB (3 bytes per pixel)
		Data size:	(width) × (height) × 3 bytes
Number of tiles	1		
Segmented processing	Supported Segmented processing can be set up in combination with processing by the CPU. Refer to the description for details.		

Description

This function has following two operation modes, and when used in combination, it is possible to outputs image normalized the histogram of the input image.

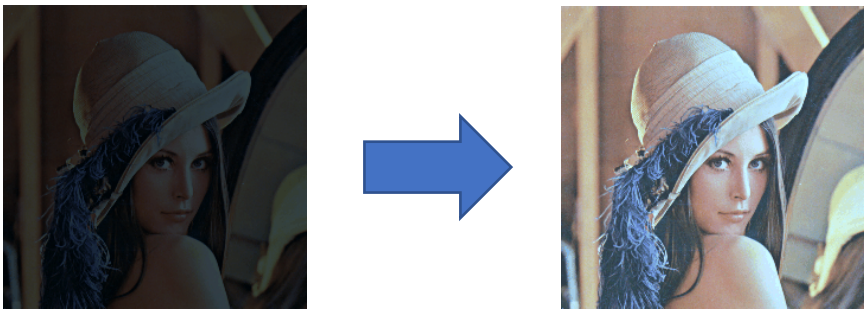
MODE1: Calculates sum and square-sum of the image at the address specified by src and outputs the result to the address specified by dst.

MODE2: Normalizes the image at the address specified by src by following calculation and outputs the result to the address specified by dst.

$$\text{output pixel value} = \{(\text{input pixel value} - \text{src\_pixel\_*\_mean}) \times \text{src\_pixel\_*\_rstd}\} \times \text{dst\_pixel\_std} + \text{dst\_pixel\_mean}$$

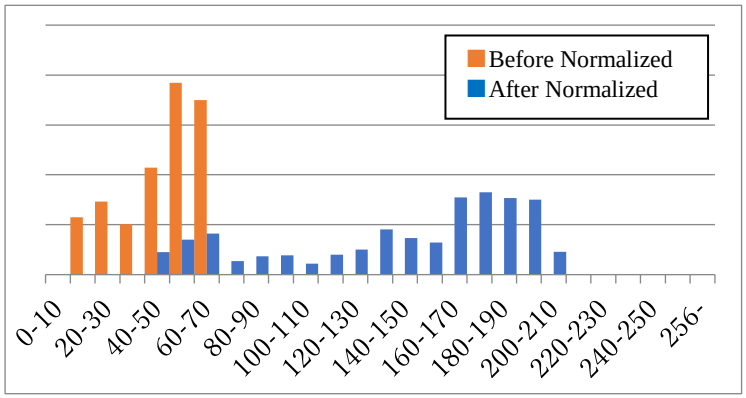
"*" is either red, green or blue.

Please set the target values of the mean and the standard deviation of the normalized image to dst_pixel_mean and dst_pixel_std. When you set dst_pixel_mean = 144 and dst_pixle_std = 48 and normalize the lower left image, this function outputs lower right image.



Input image Output image

Also following figure is the R component histogram of the input image (before normalized) and the output image (after normalized)



The R component histogram of the input image and the output image

Follow the steps below to obtain the output image normalized the histogram of the input image.
1. Calculates sum and square-sum of pixel values by executing MODE 1.
2. Calculates src_pixel_*_mean and src_pixel_*_rstd using the output result of 1. and the following equations in the CPU.
3. Outputs normalized image by executing MODE2 using the calculation results of 2. as parameters.

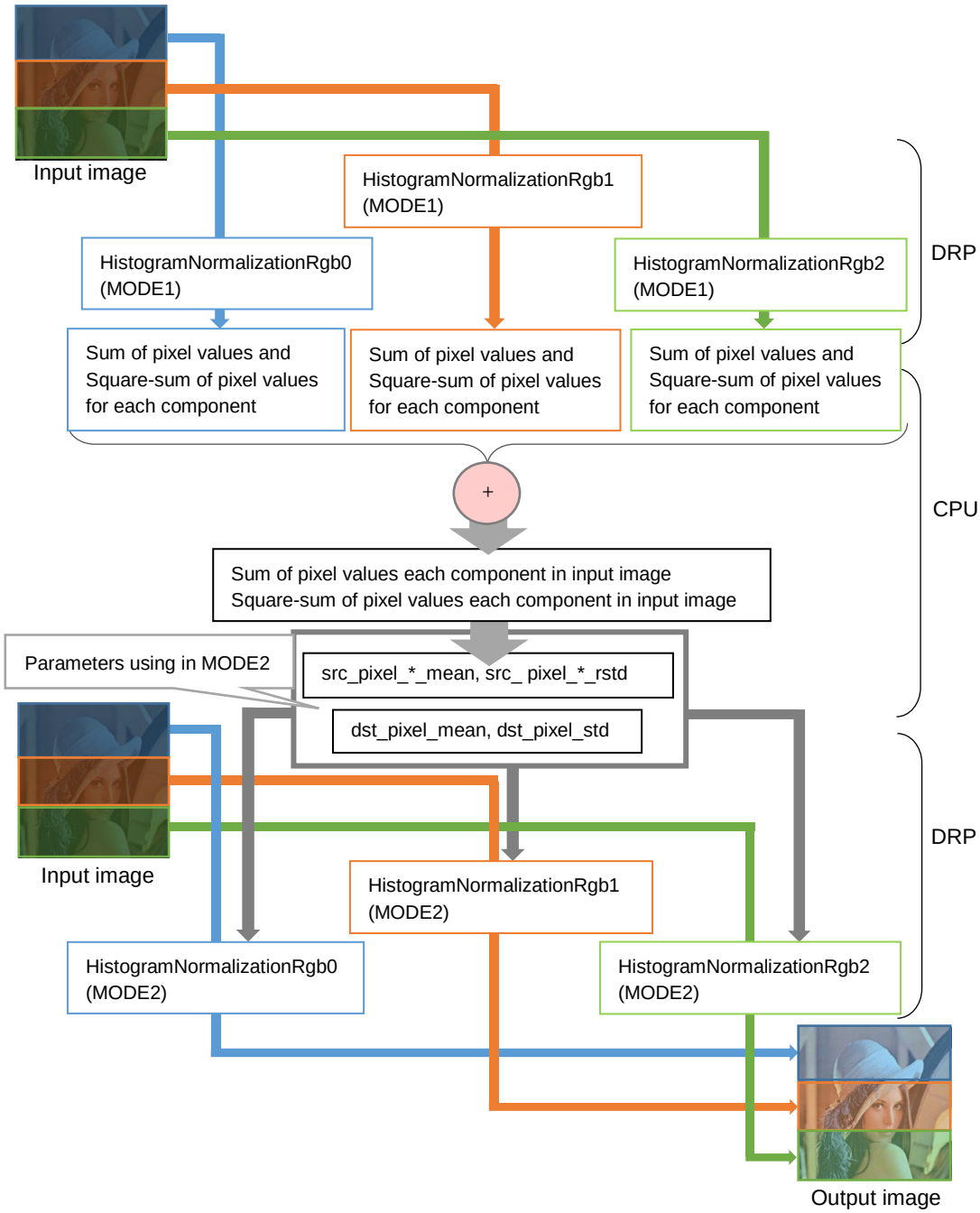
$$\text{src\_pixel\_} * \text{\_mean} = \text{sum of pixel value} \div (\text{width} \times \text{height}) \times 4096$$

$$\text{src\_pixel\_} * \text{\_rstd} = 1 \div \left(\sqrt{\text{square-sum of pixel value} \div (\text{width} \times \text{height}) - \text{src\_pixel\_} * \text{\_mean}^2}\right) \times 4096$$

Because src_pixel_*_mean and src_pixel_*_rstd are fixed point (The upper 20 bits are integer part and lower 12 bits are a decimal part), please be sure to multiply 4096 like the above equations.

This function can execute as segmented processing in combination with CPU processing. An example of three parallel processing flow is shown below.

1. The input data are segmented into three areas. Specify prescribed src, dst, width and height for HistogramNormalizationRgb of respective areas. And specify mode 1.
2. After DRP processing is complete, calculate src_pixel_*_mean and src_pixel_*_rstd using the sum and the square-sum of the pixel values in the dst area of each HistogramNormalizationRgb in the CPU.
3. The input data are segmented into three areas. Specify prescribed src, dst, width and height for HistogramNormalizationRgb of respective areas. Also, as common parameters, specify the calculation results of 2. for src_pixel_*_mean and src_pixel_*_rstd, and arbitrary values for dst_pixel_mean and dst_pixel_std, and 2 for mode.
4. After DRP processing is complete, outputs the normalized image.



This function allows the same address to be specified for both src and dst in MODE2.

Note Do not set values other than those calculated by the equations described in the Description to src_pixel_*_mean and src_pixel_*_rstd because it may cause malfunction.

4.3.15 Opening

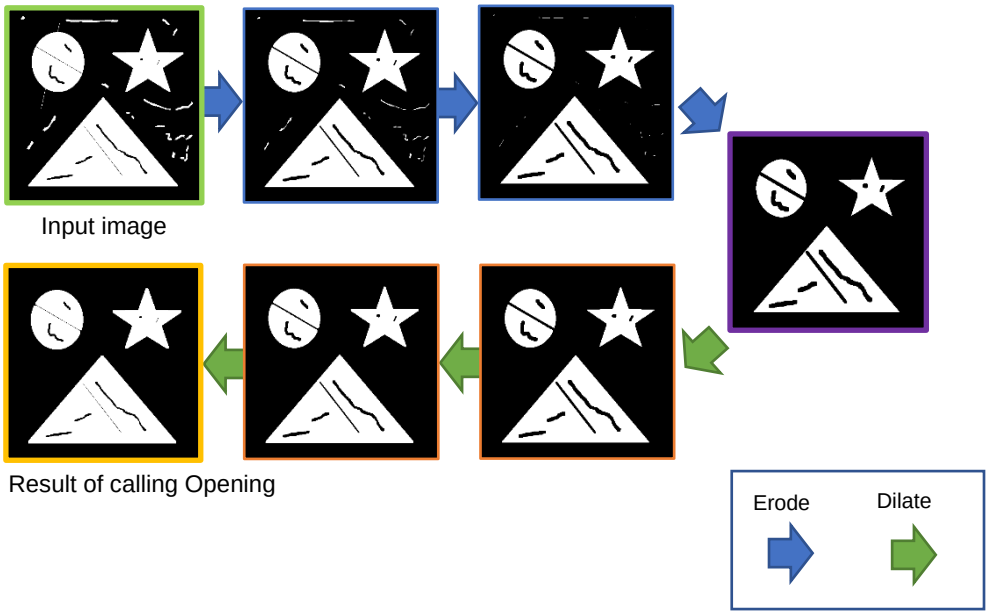
Opening

Removes noise from black portions by shrinkage (erosion) followed by expansion (dilation)

Description

Opening involves the repeated application of shrinkage (erosion) within the white parts, followed by the repeated application of expansion (dilation). The erosion and dilation are repeated the same number of times. This is useful for eliminating noise in monochrome images.

In other words, this processing involves the application of a combination of the Erode and Dilate functions of the DRP Library. Refer to the respective sections for the specifications of the Erode function and the Dilate function.



The explanation of the Opening processing is for when the number of iterations of both the Erode and Dilate functions is three.

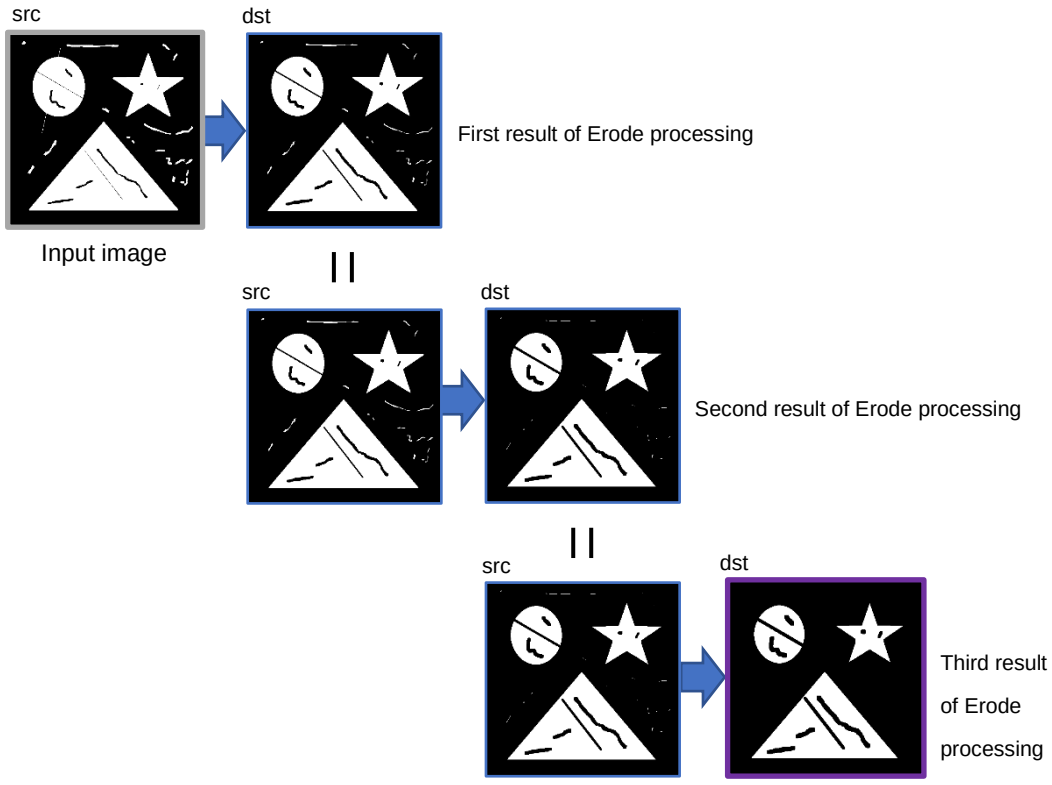
Erosion

An overview of repeating Erode processing three times is shown below.

In the first iteration of Erode processing, the image which is input is set as the input image for processing by the Erode function.

In the second iteration of Erode processing, the output image of the first iteration is set as the input image for processing by the Erode function.

In the third iteration of Erode processing, the output image of the second iteration is set as the input image for processing by the Erode function.



Dilation

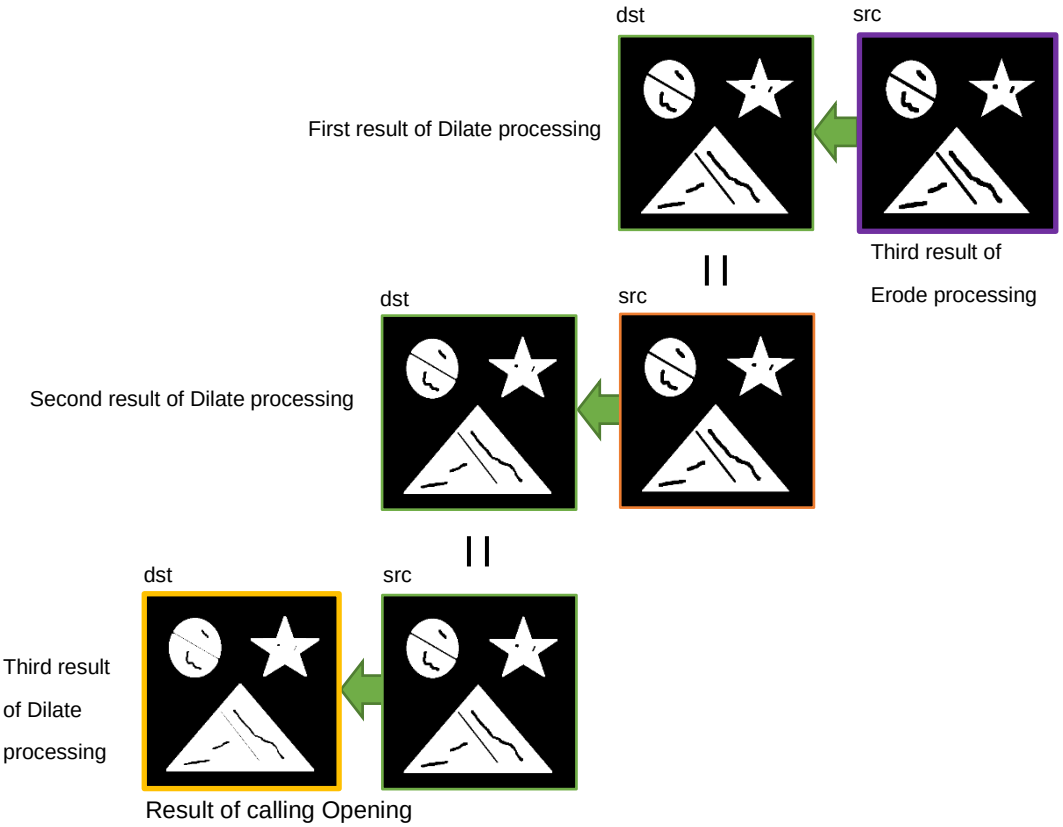
An overview of repeating Dilate processing three times is shown below.

In the first iteration of Dilate processing, the output image of the third Erode processing is set as the input image for processing by the Dilate function.

In the second iteration of Dilate processing, the output image of the first iteration is set as the input image for processing by the Dilate function.

In the third iteration of Dilate processing, the output image of the second iteration is set as the input image for processing by the Dilate function.

The output image of the third Dilate processing becomes the result image of performing Opening.



The processing performed by this function is equivalent to that of the OpenCV cv::morphologyEx function with specifying MORPH_OPEN to the argument op, cv::Mat() to kernel, Point(-1,-1) to anchor, the iteration number to iterations, and BORDER_REPLICATE to borderType.

Reference URL: https://opencv.org/

Note	If the processing of Erode and Dilate is to be segmented, only proceed with a next stage of processing after all segments of the resulting images in the current stage have been obtained.

4.3.16 Closing

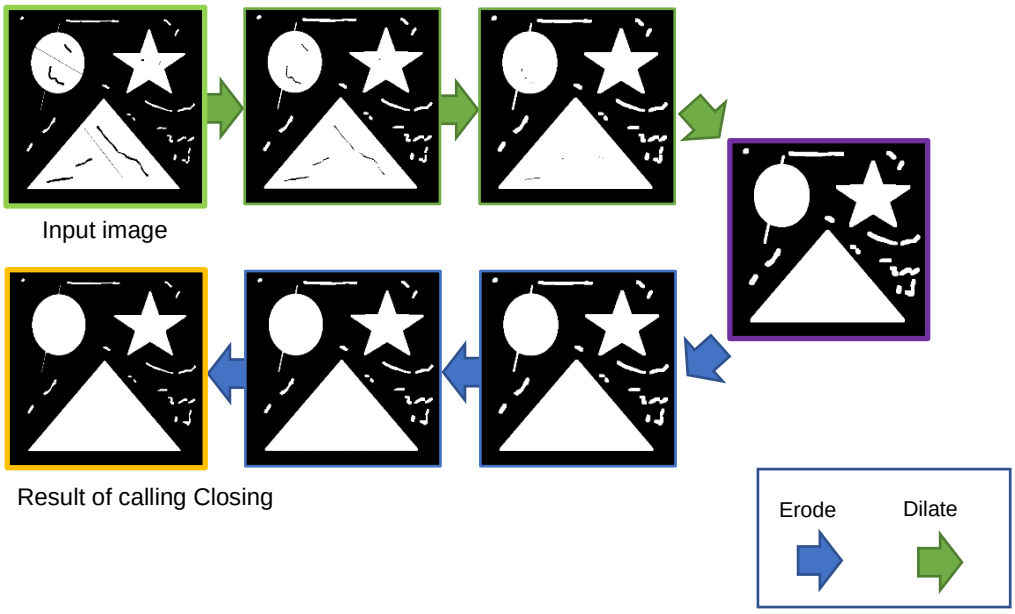
Closing

Removes noise from white portions by expansion (dilation) followed by shrinkage (erosion)

Description

Closing involves the repeated application of expansion (dilation) within the white parts, followed by the repeated application of shrinkage (erosion). The dilation and erosion are repeated the same number of times. This is useful for eliminating noise in monochrome images.

In other words, this processing involves the application of a combination of the Dilate and Erode functions of the DRP Library. Refer to the respective sections for the specifications of the Dilate function and the Erode function.



The explanation of the Closing processing is for when the number of iterations of both the Dilate and Erode functions is three.

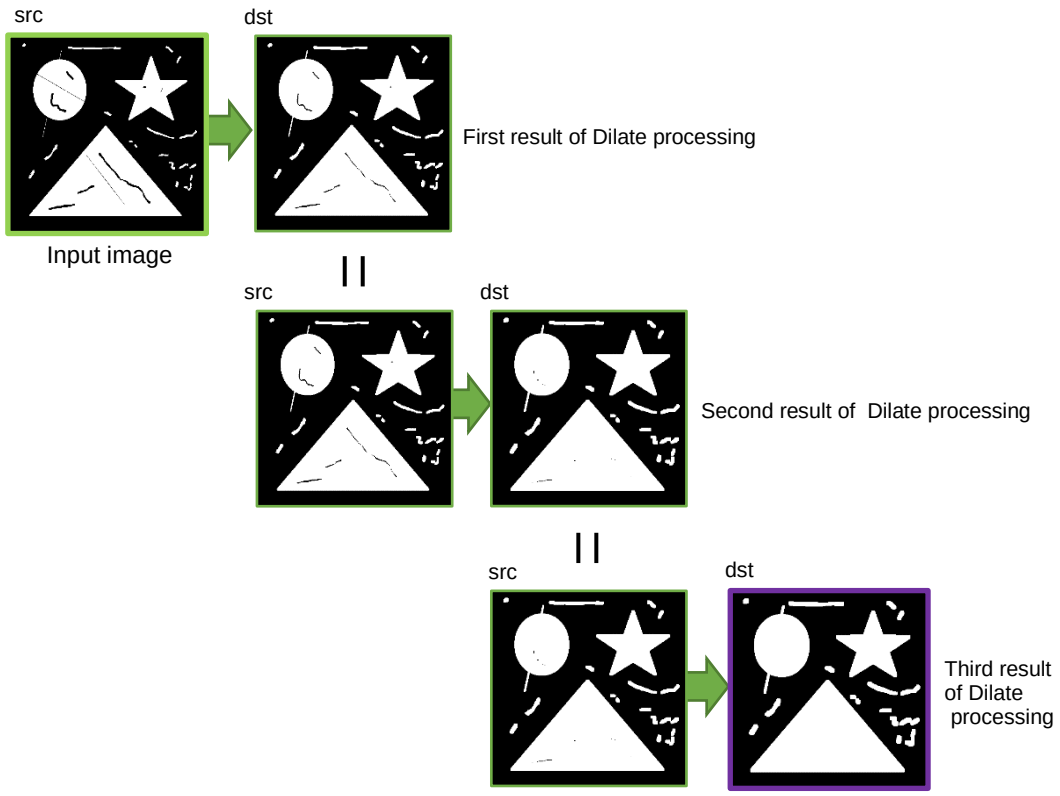
Dilation

An overview of repeating Dilate processing three times is shown below.

In the first iteration of Dilate processing, the image which is input is set as the input image for processing by the Dilate function.

In the second iteration of Dilate processing, the output image of the first iteration is set as the input image for processing by the Dilate function.

In the third iteration of Dilate processing, the output image of the second iteration is set as the input image for processing by the Dilate function.



Erosion

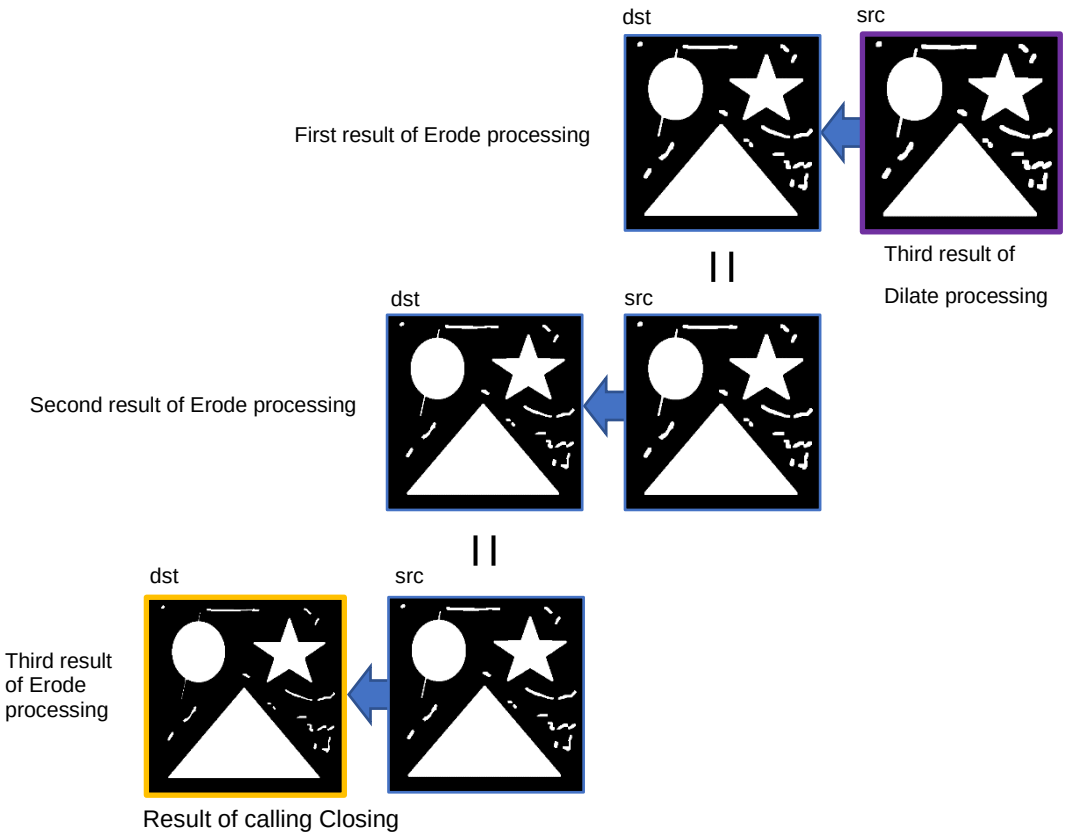
An overview of repeating Erode processing three times is shown below.

In the first iteration of Erode processing, the output image of the third Dilate processing is set as the input image for processing by the Erode function.

In the second iteration of Erode processing, the output image of the first iteration is set as the input image for processing by the Erode function.

In the third iteration of Erode processing, the output image of the second iteration is set as the input image for processing by the Erode function.

The output image of the third Erode processing becomes the result image of performing Closing.



The processing performed by this function is equivalent to that of the OpenCV cv::morphologyEx function with specifying MORPH_CLOSE to the argument op, cv::Mat() to kernel, Point(-1,-1) to anchor, the iteration number to iterations, and BORDER_REPLICATE to borderType.

Reference URL: https://opencv.org/

Note	If the processing of Dilate and Erode is to be segmented, only proceed with a next stage of processing after all segments of the resulting images in the current stage have been obtained.

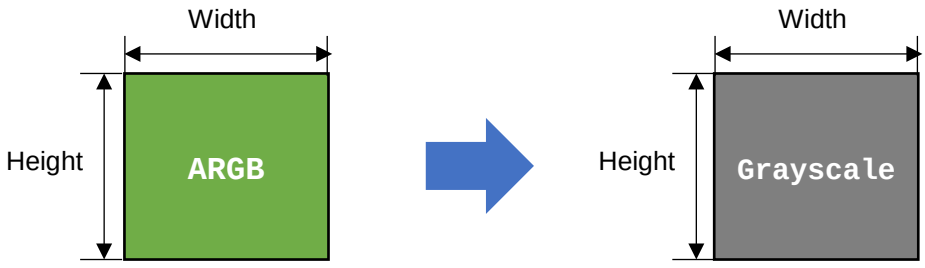
4.4 Image Conversion

4.4.1 Argb2Grayscale

Argb2Grayscale

Converts from ARGB to grayscale

Configuration data file			r_drp_argb2grayscale.dat
Supported version			0.90
Configuration data size (byte)			14368
Header file			r_drp_argb2grayscale.h
Parameter	Structure name		
	r_drp_argb2grayscale_t		
	Member name	Type	Description
	src	uint32_t	Input image address
	dst	uint32_t	Output image address
	width	uint16_t	Image width (pixels)
	height	uint16_t	Image height (pixels)
I/O details	Input image	Address:	Specified by src. (Specify an address that differs from dst.)
		Width (pixels):	Specified by width. (16 to 1280, integer multiple of 2)
		Height (pixels):	Specified by height. (1 to 960)
		Format:	ARGB (4 bytes per pixel)
		Data size:	(width) × (height) × 4 bytes
	Output image	Address:	Specified by dst. (Specify an address that differs from src.)
		Width (pixels):	Same as input image
		Height (pixels):	Same as input image
		Format:	8-bit grayscale (1 byte per pixel)
		Data size:	(width) × (height) × 1 byte
Number of tiles	1		
Segmented processing	Supported		
Description	This function converts the image at the address specified by src from ARGB format to grayscale and outputs the result to the address specified by dst.		



The function uses the following equation to convert between image formats.

$$Grayscale = (A \times 0 + R \times 16384 + G \times 40960 + B \times 8192) \div 65536$$

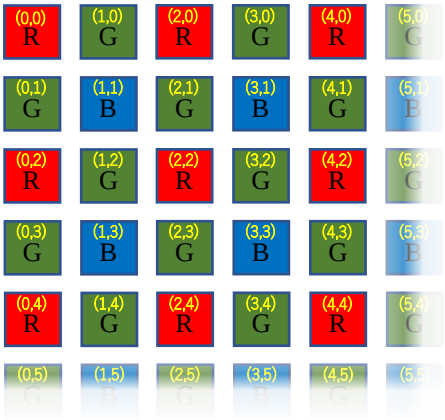
Note	None

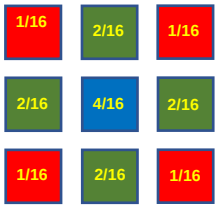
4.4.2 Bayer2Grayscale

Bayer2Grayscale

Converts from RAW data acquired from CMOS to grayscale

Configuration data file			r_drp_bayer2grayscale.dat
Supported version			0.91
Configuration data size (byte)			62912
Header file			r_drp_bayer2grayscale.h
Parameter	Structure name		
	r_drp_bayer2grayscale_t		
	Member name	Type	Description
	src	uint32_t	Input image address
	dst	uint32_t	Output image address
	width	uint16_t	Image width (pixels)
	height	uint16_t	Image height (pixels)
	top	uint8_t	1: Top edge border processing 0: No top edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the top edge of the source image, otherwise, specify 0.
	bottom	uint8_t	1: Bottom edge border processing 0: No bottom edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the bottom edge of the source image, otherwise, specify 0.

I/O details	Input image	Address:	Specified by src.
		Width (pixels):	Specified by width. (16 to 1280)
		Height (pixels):	Specified by height. (4 to 960)
		Data size:	(width) × (height) × 1 byte
		Format The input image format is as follows. When the coordinates of the upper left corner in input image are (0,0), both X and Y coordinates being even numbers represents "red," both being odd numbers represents "blue," and any other combination represents "green." This produces the Bayer array shown below.  (X coordinate, Y coordinate) = (even, even): red (even, odd): green (odd, even): green (odd, odd): blue Bayer arrays other than the above can be supported either by changing the camera settings or using the VIN function of the RZ/A2M. Refer to the description below for details.	
	Output image	Address:	Specified by dst.
		Width (pixels):	Same as input image
		Height (pixels):	Same as input image
		Format:	8-bit grayscale (1 byte per pixel)
		Data size:	(width) × (height) × 1 byte
Number of tiles	1		
Segmented processing	Supported		

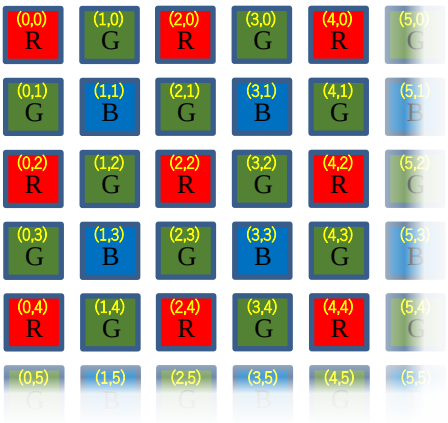
Description	This function converts the image at the address specified by src from Bayer format to 8-bit grayscale format and outputs the result to the address specified by dst. First, the function converts the input image to RGB by linear interpolation using a 3 × 3 filter. Then it converts from RGB to Y and calculates brightness values. In linear interpolation using a 3 × 3 filter, the 3 × 3 grid consists of the pixel to be converted and the pixels adjacent to it. The pixel values are multiplied by the following multipliers and the results for each color component are added up. Value of center pixel: 4/16× Values of pixels immediately above, below, left, and right: 2/16× Values of diagonally adjacent pixels: 1/16× These are then multiplied by the reciprocals of the Bayer color density values (4 for red and blue, 2 for green), to obtain the RGB values for the pixel being converted.  • **Center: 4/16×** • **Above, below, left, and right: 2/16×** • **Diagonally adjacent: 1/16×** **Each is multiplied by the respective multiplier indicated above and the results for each color component added up. These are then multiplied by the reciprocals of the color density values (4 for red and blue, 2 for green).** The following equation is used to convert from RGB to Y. Y = (Red * 76 + Green * 152 + Blue * 28) / 256 For the pixels at the left and right edges of the screen, a portion of the 3 × 3 filter grid is outside the input image area and therefore cannot be referenced. Instead, border reflection (OpenCV BORDER_REFLECT_101), in which the values of pixels 1 line further inward are referenced, is performed. Reference URL: https://opencv.org/ When top and bottom are both set to 1, equivalent border reflection is also performed at the top and bottom edges of the image. Set top and bottom to 1 if the input image is not segmented. When using a camera with a Bayer array that differs from that shown in the figure for "Input image" under "I/O details," crop and capture the image in a position such that the upper left corner is red. To crop the image, either clip the output image range of the camera or, when using an MIPI camera, clip the input image range on the RZ/A2M. For information on settings for the latter method, refer to section 48, Video Input Module, in RZ/A2M Group User's Manual: Hardware, or the description of range clipping (pre-stage) in the user's manual of the MIPI driver. This function allows specification of the same address for both src and dst as long as the processing is not segmented.
Note	None

4.4.3 Bayer2Rgb

Bayer2Rgb

Converts from RAW data acquired from CMOS to RGB

Configuration data file			r_drp_bayer2rgb.dat
Supported version			0.90
Configuration data size (byte)			92288
Header file			r_drp_bayer2rgb.h
Parameter	Structure name		
	r_drp_bayer2rgb_t		
	Member name	Type	Description
	src	uint32_t	Input image address
	dst	uint32_t	Output image address
	width	uint16_t	Image width (pixels)
	height	uint16_t	Image height (pixels)
	top	uint8_t	1: Top edge border processing 0: No top edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the top edge of the source image, otherwise, specify 0.
	bottom	uint8_t	1: Bottom edge border processing 0: No bottom edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the bottom edge of the source image, otherwise, specify 0.

I/O details	Input image	Address:	Specified by src. (Specify an address that differs from dst.)
		Width (pixels):	Specified by width. (16 to 1280, integer multiple of 2)
		Height (pixels):	Specified by height. (4 to 960, integer multiple of 2)
		Data size:	(width) × (height) × 1 byte
		Format The input image format is as follows. When the coordinates of the upper left corner in input image are (0,0), both X and Y coordinates being even numbers represents "red," both being odd numbers represents "blue," and any other combination represents "green." This produces the Bayer array shown below.  (X coordinate, Y coordinate) = (even, even): red (even, odd): green (odd, even): green (odd, odd): blue Bayer arrays other than the above can be supported either by changing the camera settings or using the VIN function of the RZ/A2M. Refer to the description below for details.	
	Output image	Address:	Specified by dst. (Specify an address that differs from src.)
		Width (pixels):	Same as input image
		Height (pixels):	Same as input image
		Format:	RGB (3 bytes per pixel)
		Data size:	(width) × (height) × 3 bytes
Number of tiles	2		
Segmented processing	Supported		

Description

This function converts the image at the address specified by src from Bayer format to RGB format and outputs the result to the address specified by dst.

This function uses adaptive color plane interpolation (ACPI) to convert the image to RGB format. This conversion method is registered under patent publication number US5629734A.

This function performs RGB conversion by ACPI as follows. The conversion of each color at each coordinate is described separately.
In the description below, the value at coordinates (x,y) in the input image is expressed as I(x,y), and the RGB values at coordinates (x,y) in the output image are expressed as R(x,y), G(x,y), and B(x,y).

1. Calculation of G Component

- If x = even number and y = odd number
- If x = odd number and y = even number

The G component in the Bayer array of the input image applies, so the input I(x,y) value is used without modification.

$$\mathrm{G}(x,y) = \mathrm{I}(x,y)$$

- If x = even number and y = even number
- If x = odd number and y = odd number

The R or B component in the Bayer array of the input image applies, so G(x,y) is calculated as follows.

First, M, which represents the degree of change in the horizontal direction, and N, which represents the degree of change in the vertical direction, are calculated.

$$\mathrm{M} = |I(x-2,y) + I(x+2,y) - 2 \times I(x,y)| + |I(x+1,y) - I(x-1,y)|$$
$$\mathrm{N} = |I(x,y-2) + I(x,y+2) - 2 \times I(x,y)| + |I(x,y+1) - I(x,y-1)|$$

Next, the two degrees of change are compared, and the interpolation value t is calculated in the direction with the smaller degree of change. (If the degrees of change are the same, the average of the interpolation values in both directions is used as interpolation value t.)

(1) If M < N

$$\mathrm{t} = \Big(2 \times \big(I(x-1,y) + I(x+1,y) + I(x,y)\big) - I(x-2,y) - I(x+2,y)\Big) \div 4$$

(2) If M > N

$$\mathrm{t} = \Big(2 \times \big(I(x,y-1) + I(x,y+1) + I(x,y)\big) - I(x,y-2) - I(x,y+2)\Big) \div 4$$

(3) If M = N

$$\mathrm{t} = \Big(2 \times \big(I(x-1,y) + I(x+1,y) + 2 \times I(x,y) + I(x,y-1) + I(x,y+1)\big) - I(x-2,y) - I(x+2,y) - I(x,y-2) - I(x,y+2)\Big) \div 8$$

The digits after the decimal point of interpolation value t are discarded, resulting in t', and the value of G(x,y) is as follows.

$$\mathrm{G}(x,y) = \begin{cases} 0, & \mathrm{t}' < 0 \\ 255, & \mathrm{t}' > 255 \\ \mathrm{t}', & 0 \le \mathrm{t}' \le 255 \end{cases}$$

2. Calculation of R Component

- If x = even number and y = even number

The R component in the Bayer array of the input image applies, so the input I(x,y) value is used without modification.

$$\mathrm{R}(x,y) = \mathrm{I}(x,y)$$

- If x = odd number and y = odd number

The B component in the Bayer array of the input image applies, so R(x,y) is calculated as follows.

First, M, which represents the degree of change in the diagonal (upper right to lower left) direction, and N, which represents the degree of change in the diagonal (upper left to lower right) direction, are calculated.

$$\mathrm{M} = |G(x+1,y-1) + G(x-1,y+1) - 2 \times G(x,y)| + |I(x-1,y+1) - I(x+1,y-1)|$$
$$\mathrm{N} = |G(x-1,y-1) + G(x+1,y+1) - 2 \times G(x,y)| + |I(x+1,y+1) - I(x-1,y-1)|$$

Next, the two degrees of change are compared, and the interpolation value t is calculated in the direction with the smaller degree of change. (If the degrees of change are the same, the average of the interpolation values in both directions is used as interpolation value t.)

(1) If M < N

$$\mathrm{t} = \Big(2 \times \big(I(x+1,y-1) + I(x-1,y+1) + G(x,y)\big) - G(x+1,y-1) - G(x-1,y+1)\Big) \div 4$$

(2) If M > N

$$\mathrm{t} = \Big(2 \times \big(I(x-1,y-1) + I(x+1,y+1) + G(x,y)\big) - G(x-1,y-1) - G(x+1,y+1)\Big) \div 4$$

(3) If M = N

$$\mathrm{t} = \Big(2 \times \big(I(x+1,y-1) + I(x-1,y+1) + 2 \times G(x,y) + I(x-1,y-1) + I(x+1,y+1)\big) - G(x+1,y-1) - G(x-1,y+1) - G(x-1,y-1) - G(x+1,y+1)\Big) \div 8$$

The digits after the decimal point of interpolation value t are discarded, resulting in t', and the value of R(x,y) is as follows.

$$\mathrm{R}(x,y) = \begin{cases} 0, & \mathrm{t}' < 0 \\ 255, & \mathrm{t}' > 255 \\ \mathrm{t}', & 0 \le \mathrm{t}' \le 255 \end{cases}$$

- If x = odd number and y = even number

The G component in the Bayer array of the input image applies, so R(x,y) is calculated as follows, taking into account the R components to the left and right in the input image and the left and right G components calculated as described in "1. Calculation of G Component."

$$\mathrm{R}(x,y) = \begin{cases} 0, & \mathrm{M} < \mathrm{N} \\ 255, & \big((M-N) >> 2\big) > 255 \\ (M-N) >> 2, & \text{Other than above} \end{cases}$$
$$\mathrm{M} = 2 \times \big(I(x-1,y) + I(x+1,y) + I(x,y)\big)$$
$$\mathrm{N} = \mathrm{G}(x-1,y) + G(x+1,y)$$

- If x = even number and y = odd number

The G component in the Bayer array of the input image applies, so R(x,y) is calculated as follows, taking into account the R components above and below in the input image and the above and below G components calculated as described in "1. Calculation of G Component."

$$\mathrm{R}(x,y) = \begin{cases} 0, & \mathrm{M} < \mathrm{N} \\ 255, & \big((M-N) >> 2\big) > 255 \\ (M-N) >> 2, & \text{Other than above} \end{cases}$$
$$\mathrm{M} = 2 \times \big(I(x,y-1) + I(x,y+1) + I(x,y)\big)$$
$$\mathrm{N} = \mathrm{G}(x,y-1) + G(x,y+1)$$

3. Calculation of B Component

- If x = odd number and y = odd number

The B component in the Bayer array of the input image applies, so the input I(x,y) value is used without modification.

$$\mathrm{B}(x,y) = \mathrm{I}(x,y)$$

- If x = even number and y = even number

The R component in the Bayer array of the input image applies, so B(x,y) is calculated as follows.

First, M, which represents the degree of change in the diagonal (upper right to lower left) direction, and N, which represents the degree of change in the diagonal (upper left to lower right) direction, are calculated.

$$\mathrm{M} = |G(x+1,y-1) + G(x-1,y+1) - 2 \times G(x,y)| + |I(x-1,y+1) - I(x+1,y-1)|$$
$$\mathrm{N} = |G(x-1,y-1) + G(x+1,y+1) - 2 \times G(x,y)| + |I(x+1,y+1) - I(x-1,y-1)|$$

Next, the two degrees of change are compared, and the interpolation value t is calculated in the direction with the smaller degree of change. (If the degrees of change are the same, the average of the interpolation values in both directions is used as interpolation value t.)

(1) If M < N

$$\mathrm{t} = \Big(2 \times \big(I(x+1,y-1) + I(x-1,y+1) + G(x,y)\big) - G(x+1,y-1) - G(x-1,y+1)\Big) \div 4$$

(2) If M > N

$$\mathrm{t} = \Big(2 \times \big(I(x-1,y-1) + I(x+1,y+1) + G(x,y)\big) - G(x-1,y-1) - G(x+1,y+1)\Big) \div 4$$

(3) If M = N

$$\mathrm{t} = \Big(2 \times \big(I(x+1,y-1) + I(x-1,y+1) + 2 \times G(x,y) + I(x-1,y-1) + I(x+1,y+1)\big) - G(x+1,y-1) - G(x-1,y+1) - G(x-1,y-1) - G(x+1,y+1)\Big) \div 8$$

The digits after the decimal point of interpolation value t are discarded, resulting in t', and the value of B(x,y) is as follows.

$$\mathrm{B}(x,y) = \begin{cases} 0, & \mathrm{t}' < 0 \\ 255, & \mathrm{t}' > 255 \\ \mathrm{t}', & 0 \le \mathrm{t}' \le 255 \end{cases}$$

- If x = even number and y = odd number

The G component in the Bayer array of the input image applies, so B(x,y) is calculated as follows, taking into account the B components to the left and right in the input image and the left and right G components calculated as described in "1. Calculation of G Component."

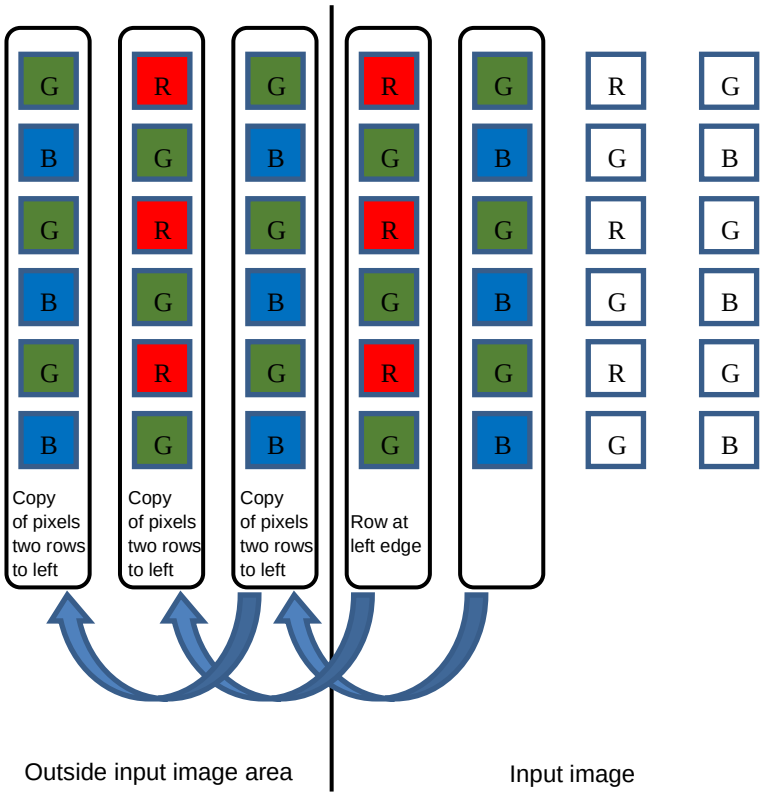
$$\mathrm{B}(x,y) = \begin{cases} 0, & \mathrm{M} < \mathrm{N} \\ 255, & \big((M-N) >> 2\big) > 255 \\ (M-N) >> 2, & \text{Other than above} \end{cases}$$
$$\mathrm{M} = 2 \times \big(I(x-1,y) + I(x+1,y) + I(x,y)\big)$$
$$\mathrm{N} = \mathrm{G}(x-1,y) + G(x+1,y)$$

- If x = odd number and y = even number

The G component in the Bayer array of the input image applies, so B(x,y) is calculated as follows, taking into account the B components above and below in the input image and the above and below G components calculated as described in "1. Calculation of G Component."

$$\mathrm{B}(x,y) = \begin{cases} 0, & \mathrm{M} < \mathrm{N} \\ 255, & \big((M-N) >> 2\big) > 255 \\ (M-N) >> 2, & \text{Other than above} \end{cases}$$
$$\mathrm{M} = 2 \times \big(I(x,y-1) + I(x,y+1) + I(x,y)\big)$$
$$\mathrm{N} = \mathrm{G}(x,y-1) + G(x,y+1)$$

When converting the pixels near the left and right edges of the screen, a portion of the data to be referred to is outside the input image area and therefore cannot be referenced. Instead, the values of the two rows of pixels at the edge are referenced, and border reflection is performed.



When top and bottom are both set to 1, equivalent border reflection is also performed at the top and bottom edges of the image. Set top and bottom to 1 if the input image is not segmented.

When using a camera with a Bayer array that differs from that shown in the figure for "Input image" under "I/O details," crop and capture the image in a position such that the upper left corner is red. To crop the image, either clip the output image range of the camera or, when using an MIPI camera, clip the input image range on the RZ/A2M. For information on settings for the latter method, refer to section 48, Video Input Module (VIN), in RZ/A2M Group User's Manual: Hardware, or the description of range clipping (pre-stage) in the user's manual of the MIPI driver.

Note	None

4.4.4 Bayer2RgbColorCorrection

Bayer2 RgbColorCorrection

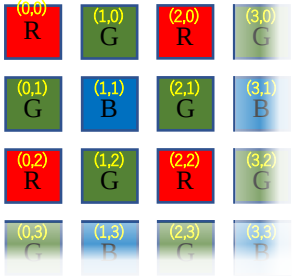
Converts from RAW data acquired from CMOS camera to RGB
(With color correction)

Configuration data file			r_drp_bayer2rgb_color_correction.dat
Supported version			0.90
Configuration data size (byte)			222656
Header file			r_drp_bayer2rgb_color_correction.h
Parameter	Structure name		
	r_drp_bayer2rgb_color_correction _t		
	Member name	Type	Description
	src	uint32_t	Input image address
	dst	uint32_t	Output image address
	width	uint16_t	Image width (pixels)
	height	uint16_t	Image height (pixels)
	gain_r	uint16_t	Gain correction value of image (R component). The upper 4 bits are an integer part, the lower 12 bits are a decimal part.
	gain_g	uint16_t	Gain correction value of image (G component). The upper 4 bits are an integer part, the lower 12 bits are a decimal part.
	gain_b	uint16_t	Gain correction value of image (B component). The upper 4 bits are an integer part, the lower 12 bits are a decimal part.
	pattern	uint8_t	Specify the bayer pattern of input image 0: RGGB 1: GRBG 2: GBRG 3: BGGR

I/O details	Input image	Address: Width (pixels): Height (pixels): Data size:	Specified by src. Specified by width. (16 to 1280) Specified by height. (4 to 960) (width) × (height) × 1 byte

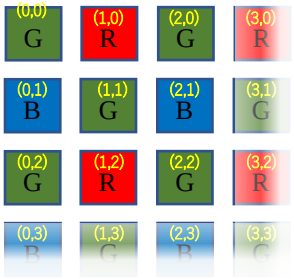
Format
The input image formats are 4 patterns shown below.

RGGB:



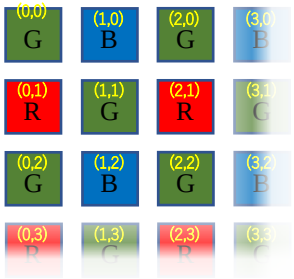
(X coordinate, Y coordinate) =
(even, even): red
(even, odd): green
(odd, even): green
(odd, odd): blue

GRBG:



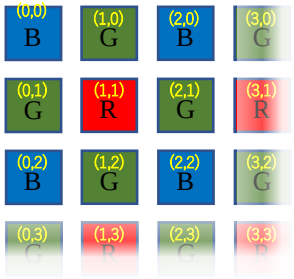
(X coordinate, Y coordinate) =
(even, even): green
(even, odd): blue
(odd, even): red
(odd, odd): green

GBRG:



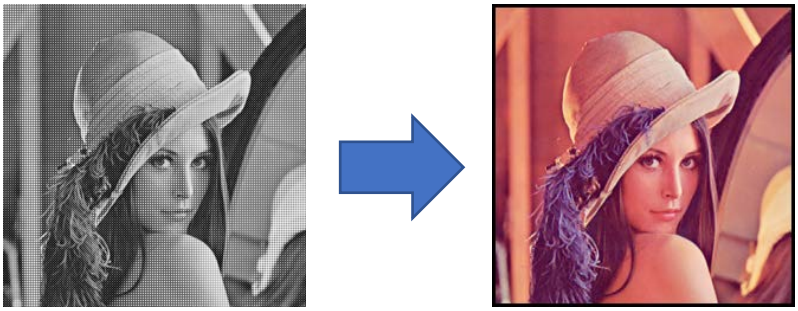
(X coordinate, Y coordinate) =
(even, even): green
(even, odd): red
(odd, even): blue
(odd, odd): green

BGGR:



(X coordinate, Y coordinate) =
(even, even): blue
(even, odd): green
(odd, even): green
(odd, odd): red

	Output image	Address: Width (pixels): Height (pixels): Format: Data size:	Specified by dst. Same as input image Same as input image RGB (3 bytes per pixel) (width) × (height) × 3 bytes
Number of tiles	6		
Segmented processing	Not supported		

Description	This function converts the image at the address specified by src from Bayer format to RGB format using Advanced Color Plane Interpolation (ACPI) and outputs the result to the address specified by dst. The ACPI is a method to obtain sharp color images by adding high frequency components to the linear interpolation value of surrounding pixels to be interpolated. This method calculates interpolation values from two directions, vertical and horizontal, then it adopts interpolation values in the direction is more continuous at the original pixel to be processed. Also, it calculates the missing component pixel using the information of other component. This function outputs black pixels at the top, bottom, left, and right 3 pixels of the output image as shown below because it does not execute border processing at the image edge.  Input image Output image This function correct respective component pixel values of RGB converted from Bayer by setting correction value to the parameter "gain_*". But, Set the value of "Actual value multiplied by 4096" to "gain_*" because it is fixed-point (the upper 4 bits are an integer part, the lower 12 bits are a decimal part). This function allows the same address to be specified for both src and dst.
Note	None

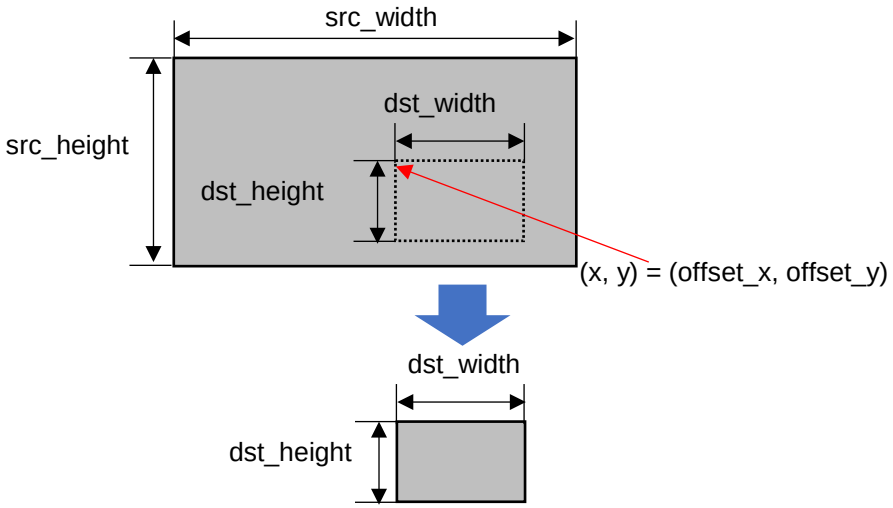
4.4.5 Cropping

Cropping

Crops a part of the image

Configuration data file			r_drp_cropping.dat	
Supported version			0.90	
Configuration data size (byte)			14688	
Header file			r_drp_cropping.h	
Parameter	Structure name			
	r_drp_cropping_t			
	Member name	Type	Description	
	src	uint32_t	Input image address	
	dst	uint32_t	Output image address	
	src_width	uint16_t	Input image width (pixels)	
	src_height	uint16_t	Input image height (pixels)	
	offset_x	uint16_t	x coordinate input image	
	offset_y	uint16_t	y coordinate input image	
	dst_width	uint16_t	Output image width (pixels)	
	dst_height	uint16_t	Output image height (pixels)	
I/O details	Input image	Address:	Specified by src.	
		Width (pixels):	Specified by src_width. (8 to 1280)	
		Height (pixels):	Specified by src_height. (8 to 960)	
		Format:	8-bit grayscale (1 byte per pixel)	
		Data size:	(src_width) × (src_height) × 1 byte	
	Output image	Address:	Specified by dst.	
		Width (pixels):	Specified by dst_width. (8 to 1280, integer multiple of 8)	
		Height (pixels):	Specified by dst_height. (8 to 960, integer multiple of 8)	
		Format:	8-bit grayscale (1 byte per pixel)	
		Data size:	(dst_width) × (dst_height) × 1 byte	
Number of tiles	1			
Segmented processing	Not supported			

Description	This function crops a rectangular portion of the size specified by the offsets from the image at the address specified by src and outputs it to the address specified by dst.



This function allows the same address to be specified for both src and dst.

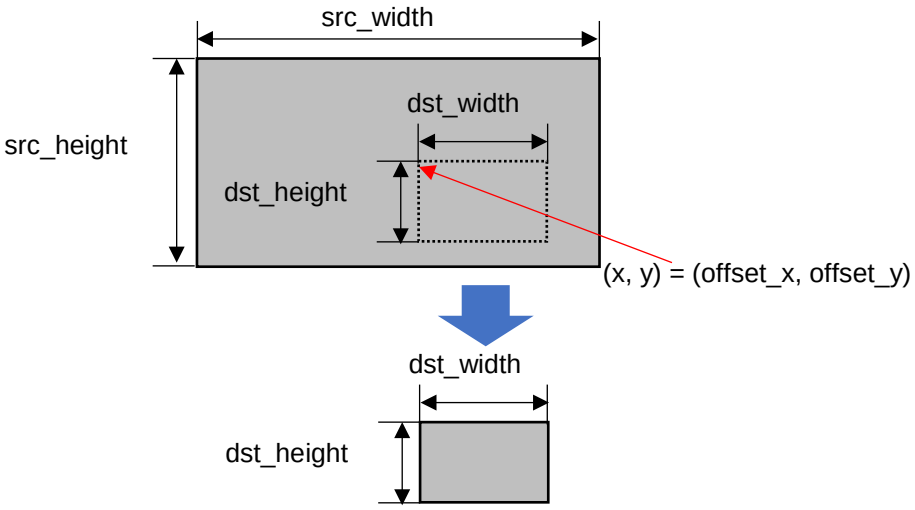
Note	The arguments should be set such that the cropped rectangular area does not extend outside of the input image area. If offset_x + dst_width exceeds src_width, or if offset_y + dst_height exceeds src_height, processing terminates with no cropping performed.

4.4.6 CroppingRgb

CroppingRgb

Crops a part of the image (RGB)

Configuration data file			r_drp_cropping_rgb.dat
Supported version			0.90
Configuration data size (byte)			20000
Header file			r_drp_cropping_rgb.h
Parameter	Structure name		
	r_drp_cropping_rgb_t		
	Member name	Type	Description
	src	uint32_t	Input image address
	dst	uint32_t	Output image address
	src_width	uint16_t	Horizontal width of input image (pixels)
	src_height	uint16_t	Vertical width of input image (pixels)
	offset_x	uint16_t	x coordinate input image
	offset_y	uint16_t	y coordinate input image
	dst_width	uint16_t	Output image width (pixels)
	dst_height	uint16_t	Output image height (pixels)
I/O details	Input image	Address:	Specified by src.
		Width (pixels):	Specified by src_width. (8 to 1280)
		Height (pixels):	Specified by src_height. (8 to 960)
		Format:	RGB (3 bytes per pixel)
		Data size:	(src_width) × (src_height) × 3 bytes
	Output image	Address:	Specified by dst.
		Width (pixels):	Specified by dst_width. (8 to 1280, integer multiple of 8)
		Height (pixels):	Specified by dst_height. (8 to 960, integer multiple of 8)
		Format:	RGB (3 bytes per pixel)
		Data size:	(dst_width) × (dst_height) × 3 bytes
Number of tiles	1		
Segmented processing	Not supported		
Description	This function crops a rectangular portion of the size specified by the offsets from the image at the address specified by src and outputs it to the address specified by dst.		



This function allows the same address to be specified for both src and dst.

Note	The arguments should be set such that the cropped rectangular area does not extend outside of the input image area. If offset_x + dst_width exceeds src_width, or if offset_y + dst_height exceeds src_height, processing terminates with no cropping performed.

4.4.7 ImageRotate

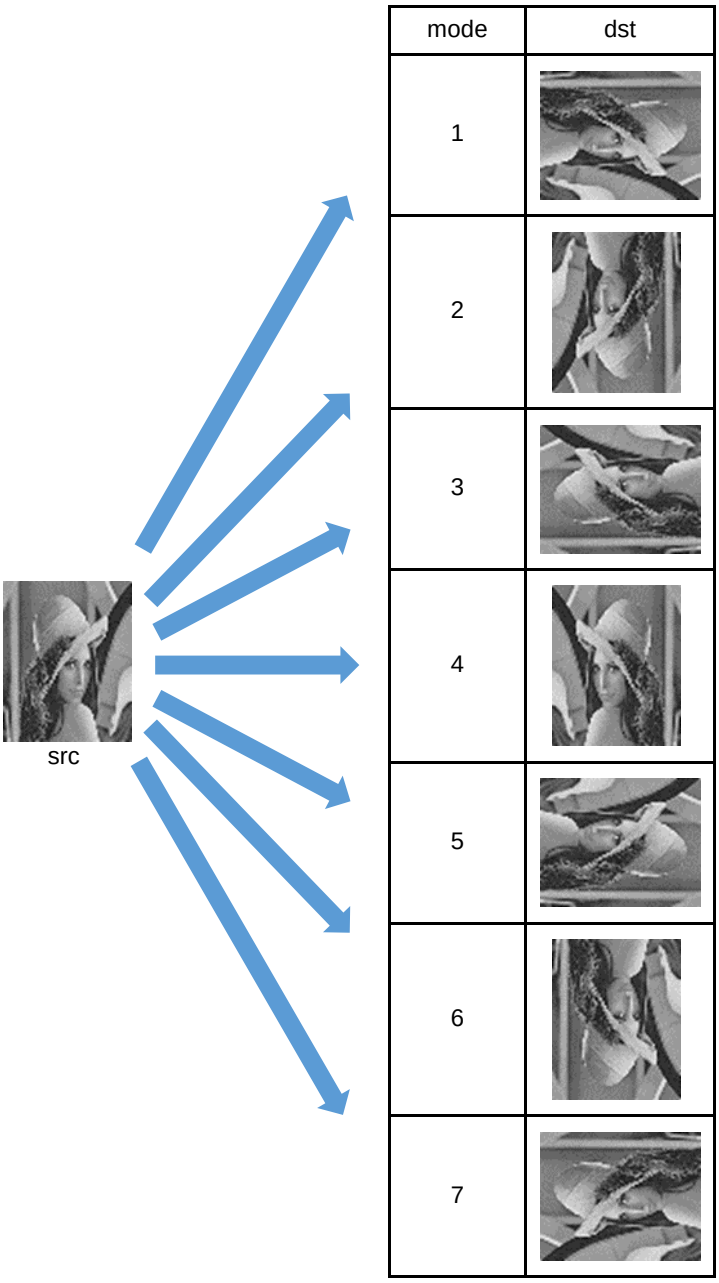
ImageRotate

Rotates the image

Configuration data file			r_drp_image_rotate.dat
Supported version			0.90
Configuration data size (byte)			56896
Header file			r_drp_image_rotate.h
Parameter	Structure name		
	r_drp_image_rotate_t		
	Member name	Type	Description
	src	uint32_t	Input image address
	dst	uint32_t	Output image address
	src_width	uint16_t	Input image width (pixels)
	src_height	uint16_t	Input image height (pixels)
	dst_stride	uint16_t	Output image stride value (0 to 1920) When set to 0, the lines of the output image are output to consecutive addresses. Specifying a value other than 0 causes the lines of the output image to be output with a spacing or "stride" between them equal to the value of this parameter. Specify either 0 or a value greater than the width of the output image. The maximum value is 1,920. Refer to the description for details.
	mode	uint8_t	1: Rotate 90° clockwise 2: Rotate 180° clockwise 3: Rotate 270° clockwise 4: Horizontal flip 5: Horizontal flip, then rotate 90° clockwise 6: Horizontal flip, then rotate 180° clockwise 7: Horizontal flip, then rotate 270° clockwise Refer to the description for details.
I/O details	Input image	Address:	Specified by src. (Specify an address that differs from dst.)
		Width (pixels):	Specified by src_width. (16 to 1280)
		Height (pixels):	Specified by src_height. (8 to 960, integer multiple of 2)
		Format:	8-bit grayscale (1 byte per pixel)
		Data size:	(src_width) × (src_height) × 1 byte
	Output image	Address:	Specified by dst. (Specify an address that differs from src.)
		Width (pixels):	Same as src_width when mode = 2, 4, or 6 Same as src_height when mode = 1, 3, 5, or 7
		Height (pixels):	Same as src_height when mode = 2, 4, or 6 Same as src_width when mode = 1, 3, 5, or 7
		Format:	8-bit grayscale (1 byte per pixel)
		Data size:	(src_width) × (src_height) × 1 byte
Number of tiles	1		
Segmented processing	Supported Refer to the description for details.		

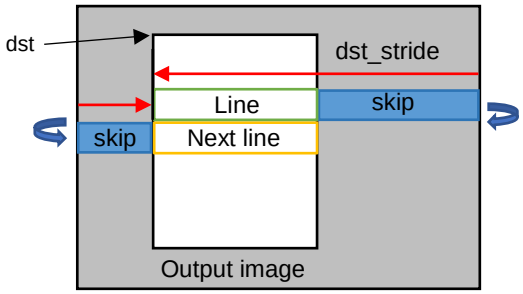
Description

This function rotates and flips as specified by mode the image at the address specified by src and outputs the result to the address specified by dst.



Ordinarily, dst_stride should be set to 0.

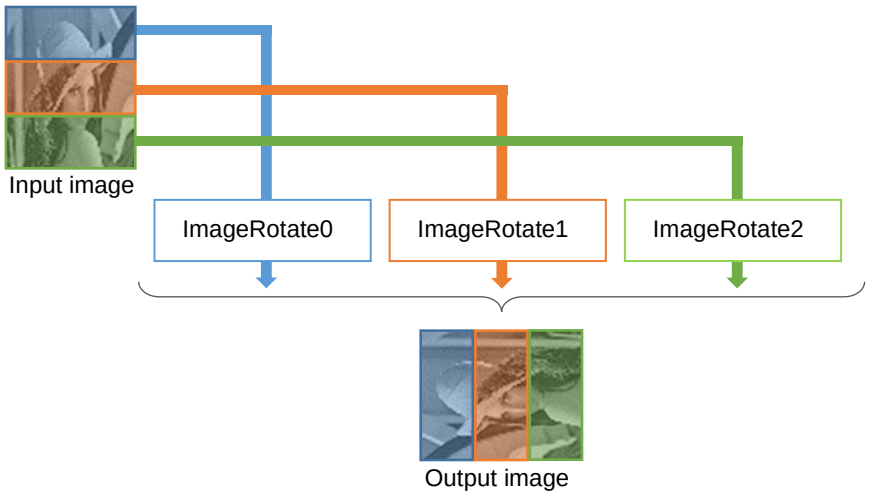
This function can be used to output an image to a partial rectangular area of a large image. To accomplish this, specify in dst_stride the number of pixels to insert between each line and the next when outputting the image.



This function can be used to perform segmented processing of an image.
In the following example, three segments are processed in parallel.

Divide the input data into three areas, ImageRotate0, ImageRotate1, and ImageRotate2, and specify specific src, dst, and src_height values for each. Use the same src_width, dst_stride, and mode values for each segment.

When mode is set to 1, 3, 5, or 7, the output image width of the three segments combined is equal to the sum of the three src_height values, so dst_stride should be set to a value equal to the sum of the three src_height values. When mode is set to a value other than the above, the output image width is equal to src_width, so dst_stride should be set to 0.



When mode = 1, this function produces results equivalent to cv::flip(src, tmp, 0); cv::transpose(tmp, dst); in OpenCV.

When mode = 2, this function produces results equivalent to cv::flip(src, dst, -1); in OpenCV.

When mode = 3, this function produces results equivalent to cv::flip(src, tmp, 1); cv::transpose(tmp, dst); in OpenCV.

When mode = 4, this function produces results equivalent to cv::flip(src, dst, 1); in OpenCV.

When mode =5, this function produces results equivalent to cv::flip(src, tmp, -1); cv::transpose(tmp, dst); in OpenCV.

When mode = 6, this function produces results equivalent to cv::flip(src, dst, 0); in OpenCV.

When mode = 7, this function produces results equivalent to cv::transpose(src, dst); in OpenCV.

Reference URL: https://opencv.org/

Note	None

4.4.8 ResizeBilinearFixed

ResizeBilinearFixed

Resizes the image (bilinear interpolation, scale factor: 2^n)

Configuration data file			r_drp_resize_bilinear_fixed.dat
Supported version			0.91
Configuration data size (byte)			138240
Header file			r_drp_resize_bilinear_fixed.h
Parameter	Structure name		
	r_drp_resize_bilinear_fixed_t		
	Member name	Type	Description
	src	uint32_t	Input image address
	dst	uint32_t	Output image address
	src_width	uint16_t	Horizontal width of input image (pixels)
	src_height	uint16_t	Vertical width of input image (pixels)
	fx	uint8_t	Horizontal scale factor The enlargement and reduction ratios are as follows. The width of the output image is 8 pixels or more.

Setting value	Enlargement/reduction ratio
0x80	0.125 (1/8)
0x40	0.25 (1/4)
0x20	0.5 (1/2)
0x10	1× (same size)
0x08	2×
0x04	4×
0x02	8×
0x01	16×

	fy	uint8_t	The vertical scale factor setting values are the same as those of fx.
I/O details	Input image	Address:	Specified by src. (Specify an address that differs from dst.)
		Width (pixels):	Specified by src_width. (128 to 1280)
		Height (pixels):	Specified by src_height. (8 to 960)
		Format:	8-bit grayscale (1 byte per pixel)
		Data size:	(src_width) × (src_height) × 1 byte
	Output image	Address:	Specified by dst. (Specify an address that differs from src.)
		Width (pixels):	Specified by src_width × (horizontal enlargement/reduction ratio)
		Height (pixels):	Specified by src_height × (vertical enlargement/reduction ratio)
		Format:	8-bit grayscale (1 byte per pixel)
		Data size:	(output image width) × (output image height) × 1 byte
Number of tiles	4		
Segmented processing	Not supported		

Description	This function enlarges or reduces the image at the address specified by src by the specified scaling factors and outputs the result to the address specified by dst. It is necessary to add or remove pixels when the image is enlarged or reduced, and this function uses bilinear method for this purpose. In the bilinear method, a grid of 2 × 2 pixels peripheral to the input image in the position corresponding to the target pixel of the output image is used and linear interpolation is applied. The processing performed by this function is equivalent to that of the OpenCV cv::resize function with specifying 0 to dsize, an enlargement/reduction ratio of 0.125 to 16 to fx and fy, and INTER_LINEAR to interpolation. Reference URL: https://opencv.org/
Note	None

4.4.9 ResizeBilinearFixedRgb

ResizeBilinearFixedRgb

Resizes the image (bilinear interpolation, Scale factor: 2^n) (RGB)

Configuration data file			r_drp_resize_bilinear_fixed_rgb.dat
Supported version			0.90
Configuration data size (byte)			202176
Header file			r_drp_resize_bilinear_fixed_rgb.h
Parameter	Structure name		
	r_drp_resize_bilinear_fixed_rgb_t		
	Member name	Type	Description
	src	uint32_t	Input image address
	dst	uint32_t	Output image address
	src_width	uint16_t	Horizontal width of input image (pixels)
	src_height	uint16_t	Vertical width of input image (pixels)
	fx	uint8_t	Horizontal scale factor The enlargement and reduction ratios are as follows. The width of the output image is 8 pixels or more.

Setting value	Enlargement/reduction ratio
0x80	0.125 (1/8)
0x40	0.25 (1/4)
0x20	0.5 (1/2)
0x10	1× (same size)
0x08	2×
0x04	4×
0x02	8×
0x01	16×

	fy	uint8_t	The vertical scale factor setting values are the same as those of fx
I/O details	Input image	Address:	Specified by src. (Specify an address that differs from dst.)
		Width (pixels):	Specified by src_width. (128 to 1280)
		Height (pixels):	Specified by src_height. (8 to 960)
		Format:	RGB (3 bytes per pixel)
		Data size:	(src_width) × (src_height) × 3 bytes
	Output image	Address:	Specified by dst. (Specify an address that differs from src.)
		Width (pixels):	Specified by src_width × (horizontal enlargement/reduction ratio)
		Height (pixels):	Specified by src_height × (vertical enlargement/reduction ratio)
		Format:	RGB (3 bytes per pixel)
		Data size:	(output image width) × (output image height) × 3 bytes
Number of tiles	6		
Segmented processing	Not supported		
Description	This function enlarges or reduces the image at the address specified by src by the specified scaling factors and outputs the result to the address specified by dst. It is necessary to add or remove pixels when the image is enlarged or reduced, and this function uses bilinear method for this purpose. In the bilinear method, a grid of 2 × 2 pixels peripheral to the input image in the position corresponding to the target pixel of the output image is used and linear interpolation is applied. The processing performed by this function is equivalent to that of the OpenCV cv::resize function with specifying 0 to dsize, an enlargement/reduction ratio of 0.125 to 16 to fx and fy, and INTER_LINEAR to interpolation. Reference URL: https://opencv.org/		
Note	None		

4.4.10 ResizeBilinear

ResizeBilinear

Resizes the image (bilinear interpolation, scale factor: any)

Configuration data file			r_drp_resize_bilinear.dat	
Supported version			0.91	
Configuration data size (byte)			379744	
Header file			r_drp_resize_bilinear.h	
Parameter	Structure name			
	r_drp_resize_bilinear_t			
	Member name	Type	Description	
	src	uint32_t	Input image address	
	dst	uint32_t	Output image address	
	src_width	uint16_t	Horizontal width of input image (pixels)	
	src_height	uint16_t	Vertical width of input image (pixels)	
	dst_width	uint16_t	Horizontal width of output image (pixels)	
	dst_height	uint16_t	Vertical width of output image (pixels)	
I/O details	Input image	Address:	Specified by src. (Specify an address that differs from dst.)	
		Width (pixels):	Specified by src_width. (32 to 1280)	
		Height (pixels):	Specified by src_height. (8 to 960)	
		Format:	8-bit grayscale (1 byte per pixel)	
		Data size:	(src_width) × (src_height) × 1 byte	
	Output image	Address:	Specified by dst. (Specify an address that differs from src.)	
		Width (pixels):	Specified by dst_width. (32 to 1280)	
		Height (pixels):	Specified by dst_height. (8 to 960)	
		Format:	8-bit grayscale (1 byte per pixel)	
		Data size:	(dst_width) × (dst_height) × 1 byte	
Number of tiles	6			
Segmented processing	Not supported			

Description

This function enlarges or reduces the image at the address specified by src and outputs the result to the address specified by dst.

It is necessary to add or remove pixels when the image is enlarged or reduced, and this function uses bilinear method for this purpose.

In the bilinear method, a grid of 2 × 2 pixels peripheral to the input image in the position corresponding to the target pixel of the output image is used and linear interpolation is applied. This function uses the following calculations for the bilinear method.

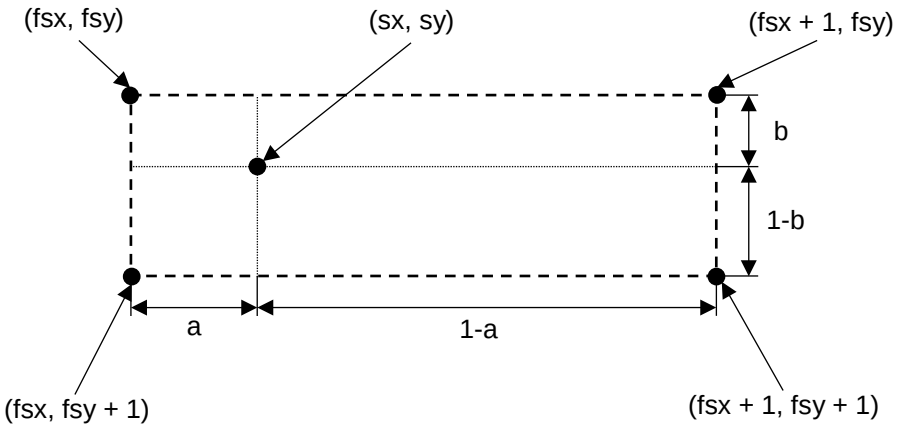
Assuming that the coordinate (sx,sy) in the input image corresponds to the coordinate (dx,dy) of the output image, sx and sy are expressed by the following equations.

$$sx = (dx + 0.5) \times src\_width \div dst\_width - 0.5$$
$$sy = (dy + 0.5) \times src\_height \div dst\_height - 0.5$$

Assuming that fsx=Floor(sx) and fsy=Floor(sy), the coordinates of the grid of 2 × 2 pixels peripheral to (sx,sy) are (fsx,fsy), (fsx+1,fsy), (fsx,fsy+1) and (fsx+1,fsy+1).

Assuming that the brightness value at the coordinate (x,y) of the input image is src(x,y) and the brightness value at the coordinate (x,y) of the output image is dst(x,y), dst(dx,dy) is expressed by the following equation.

$$dst(dx, dy) = (1 - b) \times (1 - a) \times src(fsx, fsy) + (1 - b) \times a \times src(fsx + 1, fsy)$$
$$+b \times (1 - a) \times src(fsx, fsy + 1) + b \times a \times src(fsx + 1, fsy + 1)$$
$$\text{However}, a = sx - fsx, b = sy - fsy$$



The processing performed by this function is equivalent to that of the OpenCV cv::resize function with specifying dst_width to the argument dsize.width, dst_height to dsize.height, and INTER_LINEAR to interpolation.

Reference URL: https://opencv.org/

Note

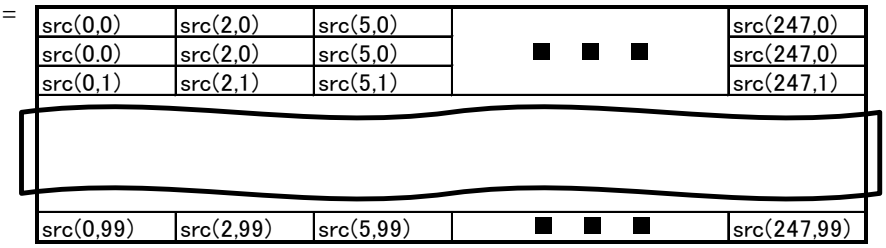
None

4.4.11 ResizeNearest

ResizeNearest

Resizes the image (nearest neighbor interpolation, scale factor: any)

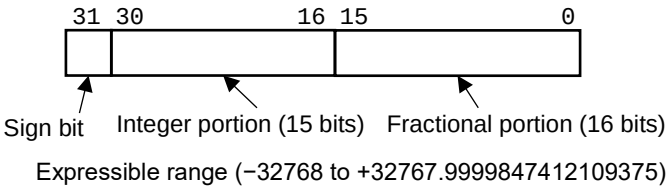
Configuration data file			r_drp_resize_nearest.dat	
Supported version			0.90	
Configuration data size (byte)			303456	
Header file			r_drp_resize_nearest.h	
Parameter	Structure name			
	r_drp_resize_nearest_t			
	Member name	Type	Description	
	src	uint32_t	Input image address	
	dst	uint32_t	Output image address	
	src_width	uint16_t	Horizontal width of input image (pixels)	
	src_height	uint16_t	Vertical width of input image (pixels)	
	dst_width	uint16_t	Horizontal width of output image (pixels)	
	dst_height	uint16_t	Vertical width of output image (pixels)	
I/O details	Input image	Address:	Specified by src. (Specify an address that differs from dst.)	
		Width (pixels):	Specified by src_width. (32 to 1280)	
		Height (pixels):	Specified by src_height. (8 to 960)	
		Format:	8-bit grayscale (1 byte per pixel)	
		Data size:	(src_width) × (src_height) × 1 byte	
	Output image	Address:	Specified by dst. (Specify an address that differs from src.)	
		Width (pixels):	Specified by dst_width. (32 to 1280)	
		Height (pixels):	Specified by dst_height. (8 to 960)	
		Format:	8-bit grayscale (1 byte per pixel)	
		Data size:	(dst_width) × (dst_height) × 1 byte	
Number of tiles	6			
Segmented processing	Not supported			

Description	This function enlarges or reduces the image at the address specified by src and outputs the result to the address specified by dst. Assuming that the brightness value at the coordinate (x,y) of the input image is src(x,y), the brightness value dst(dx,dy) at the coordinate (dx,dy) of the output image is expressed by the following equation. $$dst(dx, dy) = src(dx \times src\_width \div dst\_width, dy \times src\_height \div dst\_height)$$ The coordinate values are truncated after the decimal point. The following figure shows an example of the output image when the size of the input image is 250 x 100 and that of the output image is 100 x 200. dst(dx,dy) =  The processing performed by this function is equivalent to that of the OpenCV cv::resize function with specifying dst_width to the argument dsize.width, dst_height to dsize.height, and INTER_NEAREST to interpolation. Reference URL: https://opencv.org/
Note	None

4.4.12 Affine

Affine

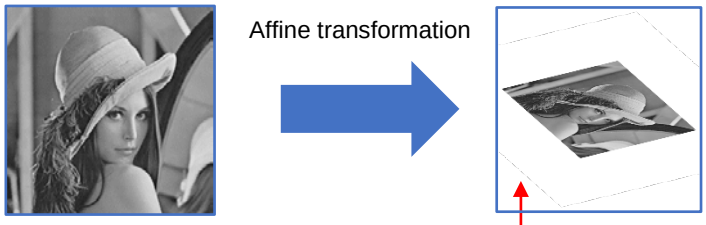
Performs parallel translation and linear transformation on the image

Configuration data file			r_drp_affine.dat
Supported version			0.90
Configuration data size (byte)			589792
Header file			r_drp_affine.h
Parameter	Structure name		
	r_drp_affine_t		
	Member name	Type	Description
	src	uint32_t	Input image address
	dst	uint32_t	Output image address
	src_width	uint16_t	Input image width (pixels)
	src_height	uint16_t	Input image height (pixels)
	dst_width	uint16_t	Output image width (pixels)
	dst_height	uint16_t	Output image height (pixels)
	m_11	int32_t	Value of element at column 1, row 1 of the transform matrix converted to fixed-point format The fixed-point format is shown below.  Expressible range (−32768 to +32767.9999847412109375) (Refer to the description for details.)
	m_12	int32_t	Value of element at column 1, row 2 of the transform matrix converted to fixed-point format The fixed-point format is the same as that of m_11. (Refer to the description for details.)
	m_13	int32_t	Value of element at column 1, row 3 of the transform matrix converted to fixed-point format The fixed-point format is the same as that of m_11. (Refer to the description for details.)
	m_21	int32_t	Value of element at column 2, row 1 of the transform matrix converted to fixed-point format The fixed-point format is the same as that of m_11. (Refer to the description for details.)
	m_22	int32_t	Value of element at column 2, row 2 of the transform matrix converted to fixed-point format The fixed-point format is the same as that of m_11. (Refer to the description for details.)
	m_23	int32_t	Value of element at column 2, row 3 of the transform matrix converted to fixed-point format The fixed-point format is the same as that of m_11. (Refer to the description for details.)
	border_value	uint8_t	Output value when outside range of referenced input image

I/O details	Input image	Address:	Specified by src. (Specify an address that differs from dst.)
		Width (pixels):	Specified by src_width. (32 to 1280)
		Height (pixels):	Specified by src_height. (8 to 960)
		Format:	8-bit grayscale (1 byte per pixel)
		Data size:	(src_width) × (src_height) × 1 byte
	Output image	Address:	Specified by dst. (Specify an address that differs from src.)
		Width (pixels):	Specified by dst_width. (32 to 1280)
		Height (pixels):	Specified by dst_height. (8 to 960)
		Format:	8-bit grayscale (1 byte per pixel)
		Data size:	(dst_width) × (dst_height) × 1 byte
Number of tiles	6		
Segmented processing	Not supported		

Description

This function performs affine transformation on the image at the address specified by src and outputs the result to the address specified by dst.



When transform matrix M is defined as follows,

$$M = \begin{pmatrix} m\_11 & m\_12 & m\_13 \\ m\_21 & m\_22 & m\_23 \\ 0 & 0 & 1 \end{pmatrix}$$

coordinates (sx,sy) of the input image are mapped to coordinates (dx,dy) expressed in the formula below by the affine transformation.

$$\begin{pmatrix} dx \\ dy \\ 1 \end{pmatrix} = M \begin{pmatrix} sx \\ sy \\ 1 \end{pmatrix}$$

For example, when enlarging or reducing the image after rotating it, and if the central coordinates of the input image during rotation are (cx,cy), the rotation angle is θ (counterclockwise rotation is the forward direction), and the image magnification is s, transform matrix M is as follows.

$$M = \begin{pmatrix} s \times \cos\theta & s \times \sin\theta & (1 - s \times \cos\theta) \times cx - s \times \sin\theta \times cy \\ -s \times \sin\theta & s \times \cos\theta & s \times \sin\theta \times cx + (1 - s \times \cos\theta) \times cy \\ 0 & 0 & 1 \end{pmatrix}$$

In addition, it is possible to create the 2 × 3 transform matrix in the box below by using OpenCV functions such as cv::getRotationMatrix2D and cv::getAffineTransform.

$$M = \begin{pmatrix} m\_11 & m\_12 & m\_13 \\ m\_21 & m\_22 & m\_23 \\ 0 & 0 & 1 \end{pmatrix}$$

Function cv::getRotationMatrix2D creates a transform matrix that rotates the image. Function cv::getAffineTransform creates a transform matrix that maps three specified points in the input and output images.

Reference URL: https://opencv.org/

This function calculates the input image coordinates through reverse transformation of the coordinates of the output image, then performs affine transformation. If M^{-1} is the reverse matrix of matrix M, the reverse transformation can be calculated as follows.

$$\begin{pmatrix} sx \\ sy \\ 1 \end{pmatrix} = M^{-1} \begin{pmatrix} dx \\ dy \\ 1 \end{pmatrix}$$

When calculating the output image, the function performs bilinear interpolation referencing the surrounding four pixels. For details of bilinear interpolation, refer to the description of ResizeBilinear.

The results produced by this function are equivalent to those of the OpenCV cv::warpAffine function with parameter M set to a transform matrix corresponding to affine transformation, dsize.width equal to dst_width, dsize.height equal to dst_height, flags set to INTER_LINEAR, borderMode set to BORDER_CONSTANT, and borderValue equal to border_value.

Reference URL: https://opencv.org/

Note

It is not possible to calculate reverse matrix M^{-1} if the transform matrix M parameters are set such that m_11 × m_22 = m_12 × m_21, so in this case border_value is output to the entire output image area.

4.5 Feature Detection

4.5.1 CannyCalculate

CannyCalculate

Canny edge detection

Configuration data file			r_drp_canny_calculate.dat	
Supported version			0.90	
Configuration data size (byte)			126080	
Header file			r_drp_canny_calculate.h	
Parameter	Structure name			
	r_drp_canny_calculate_t			
	Member name	Type	Description	
	src	uint32_t	Input image address	
	dst	uint32_t	Output image address	
	width	uint16_t	Image width (pixels)	
	height	uint16_t	Image height (pixels)	
	work	uint32_t	Work area address	
	threshold_high	uint8_t	Edge upper limit determination value ((threshold_low + 1) to 255)	
	threshold_low	uint8_t	Edge lower limit determination value (0 to (threshold_high – 1))	
	top	uint8_t	1: Top edge border processing 0: No top edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the top edge of the source image, otherwise, specify 0.	
	bottom	uint8_t	1: Bottom edge border processing 0: No bottom edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the bottom edge of the source image, otherwise, specify 0.	
I/O details	Input image	Address:	Specified by src.	
		Width (pixels):	Specified by width. (16 to 1280, integer multiple of 16)	
		Height (pixels):	Specified by height. (5 to 960)	
		Format:	8-bit grayscale (1 byte per pixel)	
		Data size:	(width) × (height) × 1 byte	
	Output image	Address:	Specified by dst.	
		Width (pixels):	Same as input image	
		Height (pixels):	Same as input image	
		Format:	8-bit edge candidates (3 categories: 0, 1, and 2) 0: Non-edge 1: Weak edge 2: Strong edge (1 byte per pixel)	
		Data size:	(width) × (height) × 1 byte	
	Work area	Address:	Specified by work.	
		Data size:	(((width) × (height + 2)) × 2) bytes	
		Description		
		The area used to store edge strength and edge direction data. Refer to the explanation below for more on edge strength and edge direction.		

Number of tiles	2
Segmented processing	Supported
Description	This function uses the Canny method to find edge candidates in the image at the address specified by src and outputs the result to the address specified by dst.

Canny edge detection produces few edge detection errors. It is also capable of outputting edges as thin lines. Canny edge detection consists of the following processing steps, performed in the order shown:

1. Noise is eliminated (Gaussian filter).
2. The edge strength and degree of accuracy is calculated, non-maximum values are suppressed, and the edges are classified.
3. Edges are determined by hysteresis threshold processing.

The OpenCV cv::Canny() function performs all of the above processing. This library produces similar edge output by using the GaussianBlur function for step 1, the CannyCalculate function for step 2, and the CannyHysterisis function for step 3.

Reference URL: https://opencv.org/

The edge candidates output by the function fall into 3 categories based on edge strength: non-edge, weak edge, and strong edge. The thresholds for determining weak edges and strong edges are set by the threshold_low and threshold_high parameters. The lower the thresholds, the larger the number of edge candidates.

Input image

Output image
threshold_low=0x18
threshold_high=0x30

Output image
threshold_low=0x05
threshold_high=0x28

Display characteristics used:
Gray: Weak edge
White: Strong edge

The function calculates the edge strength and direction as described below.

$$\mathrm{Gx} = \begin{bmatrix} G_{00} & G_{01} & G_{02} \\ G_{10} & G_{11} & G_{12} \\ G_{20} & G_{21} & G_{22} \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\mathrm{Gy} = \begin{bmatrix} G_{00} & G_{01} & G_{02} \\ G_{10} & G_{11} & G_{12} \\ G_{20} & G_{21} & G_{22} \end{bmatrix} \times \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Edge strength = $((Gx)^2 + (Gy)^2) >> 7$

```
if ( 3 * abs(Gx) <= 8 * abs(Gy))        // 21 degrees or less
        Edge direction = DIR0
else if (20 * abs(Gx) > 8 * abs(Gy))  // More than 67 degrees
        Edge direction = DIR90
else
        Edge direction = (sign(Gx)==sign(Gy)) ? DIR45 : DIR135
```

Note	None

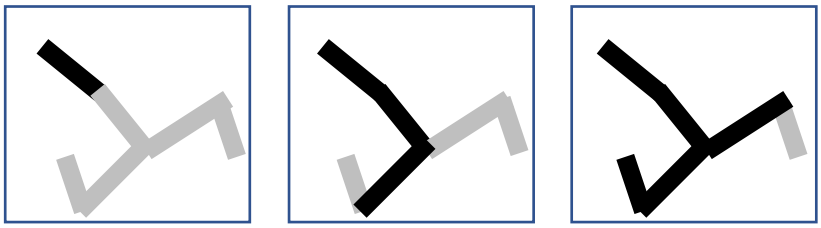
4.5.2 CannyHysterisis

CannyHysterisis

Threshold processing using hysteresis

Configuration data file			r_drp_canny_hysterisis.dat
Supported version			0.90
Configuration data size (byte)			358752
Header file			r_drp_canny_hysterisis.h
Parameter	Structure name		
	r_drp_canny_hysterisis_t		
	Member name	Type	Description
	src	uint32_t	Input image address
	dst	uint32_t	Output image address
	width	uint16_t	Image width (pixels)
	height	uint16_t	Image height (pixels)
	work	uint32_t	Work area address
	iterations	uint8_t	Maximum number of iterations (1 to 254) Infinite number of iterations (255)
I/O details	Input image	Address:	Specified by src.
		Width (pixels):	Specified by width. (16 to 1280, integer multiple of 8)
		Height (pixels):	Specified by height. (16 to 960, integer multiple of 4)
		Format:	Edge candidate (3 values: 0, 1, or 2) 0: Non-edge 1: Weak edge 2: Strong edge (1 byte per pixel)
		Data size:	(width) × (height) × 1 byte
	Output image	Address:	Specified by dst.
		Width (pixels):	Same as input image
		Height (pixels):	Same as input image
		Format:	Detected edge (2 values: 0 or 255) 0: Non-edge 255: Edge (1 byte per pixel)
		Data size:	(width) × (height) × 1 byte
	Work area	Address:	Specified by work.
		Data size:	(width) × (height) × 1 byte
		Description	
		The area used to store data during hysteresis processing.	

Number of tiles	6
Segmented processing	Not supported
Description	This function performs hysteresis threshold processing on the image (edge candidates) at the address specified by src and outputs the resulting edge image to the address specified by dst. (Edge detection using the Canny method is the second part of the processing. For details, refer to the description of the CannyCalculate function.) In hysteresis threshold processing the input edge candidates are checked, each weak edge that is connected to a strong edge is output as an edge, and each weak edge that is not connected to a strong edge is output as a non-edge. Checking is performed to confirm connections both above and below. When a weak edge is determined to be an edge, any weak edge connected to that edge must also be checked, so the processing is repeated up to the maximum number of iterations. If search continue twice that do not change to strong edge continue, the process ends. (The processing time and accuracy should be considered when choosing the setting value.)



Input image | 1st search (search below) | 2nd search (search above) | ... Maximum number of iterations

Display characteristics used:

Gray: Weak edge

Black: Strong edge

Weak edge changed to strong edge, so continue.

Weak edge changed to strong edge, so continue.

Input image | Output image

Display characteristics used:

Gray: Weak edge

White: Strong edge

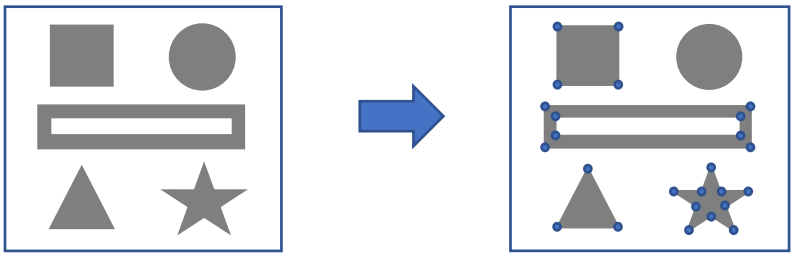
Note	None

4.5.3 CornerHarris

CornerHarris

Detects the corner contained in the image using the method devised by Chris Harris

Configuration data file			r_drp_corner_harris.dat
Supported version			0.90
Configuration data size (byte)			353088
Header file			r_drp_corner_harris.h
Parameter	Structure name		
	r_drp_corner_harris_t		
	Member name	Type	Description
	src	uint32_t	Input image address
	dst	uint32_t	Output image address Stores the response of the Harris detector.
	width	uint16_t	Image width (pixels)
	height	uint16_t	Image height (pixels)
	shift	uint8_t	Harris detector response right-shift amount This function right-shifts the 32-bit Harris detector response by the amount specified by this argument, and outputs the result as the saturation calculation with a value from 0 to 255. Since Harris detector response values are often in the range from 256 to 65,535, a setting value is 8 is recommended.
I/O details	Input image	Address:	Specified by src.
		Width (pixels):	Specified by width. (16 to 1280)
		Height (pixels):	Specified by height. (8 to 960)
		Format:	8-bit grayscale (1 byte per pixel)
		Data size:	(width) × (height) × 1 byte
	Output image (Harris detector response)	Address:	Specified by dst.
		Width (pixels):	Same as input image
		Height (pixels):	Same as input image
		Format:	Vertex detection result (0 to 255) The larger the value, the greater the likelihood of a vertex. (1 byte per pixel)
		Data size:	(width) × (height) × 1 byte

Number of tiles	6
Segmented processing	Not supported
Description	This function applies a Harris detector to the image at the address specified by src, detects vertexes within the image, and outputs the result to the address specified by dst. The Harris detector recognizes vertexes by identifying cases where the characteristics of the immediate vicinity of the target pixel differ from the characteristics of the periphery.  A simplified representation of detection of vertexes in the input image The calculations performed by the Harris detector are as follows. The sum of the slopes in the entirety of the 3 × 3 pixel adjacent area is calculated to obtain a 2 × 2 slope distribution matrix ($M^{(x,y)}$) for the target pixel. Then the following feature value is calculated. $$\mathrm{dst}(x,y) = \det M^{(x,y)} - k\left(\mathrm{tr}M^{(x,y)}\right)^2$$ The intrinsic coefficient of corner detection quantity is represented as k, and experience shows that a value of 0.04 is 0.15 is good. This function uses a value of 0.0625. The processing performed by this function is equivalent to that of the OpenCV cv::conerHarris function with the specification of 3 for blockSize argument, 3 for apertureSize, 0.0625 for k, and BORDER_REFLECT_101 for borderType. Reference URL: https://opencv.org/ This function allows the same address to be specified for both src and dst.
Note	None

4.5.4 CircleFitting

CircleFitting

Detects circle from the input image

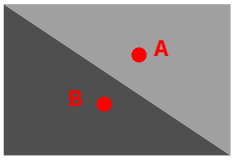
Configuration data file			r_drp_circle_fitting.dat	
Supported version			0.90	
Configuration data size (byte)			160160	
Header file			r_drp_circle_fitting.h	
Parameter	Structure name			
	r_drp_circle_fitting_t			
	Member name	Type	Description	
	src	uint32_t	Input image address	
	dst	uint32_t	Output data address	
	src_width	uint16_t	Input image width (pixels)	
	src_height	uint16_t	Input image height (pixels)	
	work	uint32_t	Work area address	
	c_area_startx	uint16_t	x-coordinate of the position from which to start searching for the center of a circle in the search area	
	c_area_starty	uint16_t	y-coordinate of the position from which to start searching for the center of a circle in the search area	
	c_area_width	uint16_t	Width (pixel) of the area in which to search for the center of a circle	
	c_area_height	uint16_t	Height (pixel) of the area in which to search for the center of a circle	
	min_radius	uint16_t	Minimum value of the radius of the circle (2 to 478) Set a value greater than the value of step.	
	max_radius	uint16_t	Maximum value of the radius of the circle (2 to 478) Set a value no less than the value of min_radius.	
	step	uint8_t	Search execution unit (pixels) in the x direction, y direction, and radial direction (1 to 51)	
I/O details	Input image	Address:	Specified by src. (Specify an address that differs from dst or work.)	
		Width (pixels):	Specified by src_width. (16 to 1280)	
		Height (pixels):	Specified by src_height. (16 to 960)	
		Format:	8-bit grayscale (1 byte per pixel)	
		Data size:	(src_width) × (src_height) × 1 byte	

	Search area	x-coordinate of the start position for searching: Specified by c_area_startx (min_radius + step to src_width - 1 - min_radius - step) y-coordinate of the start position for searching: Specified by c_area_starty (min_radius + step to src_height - 1 - min_radius - step) Width (pixels): Specified by c_area_width. (1 to src_width - c_area_startx - min_radius - step) Height (pixels): Specified by c_area_height. (1 to src_height - c_area_starty - min_radius - step) Description The search area of the input image in which to search for the center of the circle Make settings such that the value of c_area_startx + c_area_width is from min_radius + step + 1 to src_width - min_radius - step. Make settings such that the value of c_area_starty + c_area_height is from min_radius + step + 1 to src_height - min_radius - step. Refer to the description for details.
	Output data	Address: Specified by dst. (Specify an address that differs from src or work.) Format: From the top address, specifications are made in the following order. x-coordinate (2 bytes) of the center of the circle that was found. y-coordinate (2 bytes) of the center of the circle that was found. Radius (2 bytes) of the circle that was found. score (2 bytes) for the circle that was found. Refer to the description for details. Data size: 8 bytes
	Work area	Address: Specified by work. (Specify an address that differs from src or dst.) Data size: (c_area_width) × ((c_area_height ÷ step)[rounded up after the decimal point]) × 6 bytes Description The area used to store data during the circle fitting processing.
Number of tiles	2	
Segmented processing	Not supported However, segmented processing can be set up in combination with processing by the CPU. Refer to the description for details.	

Description

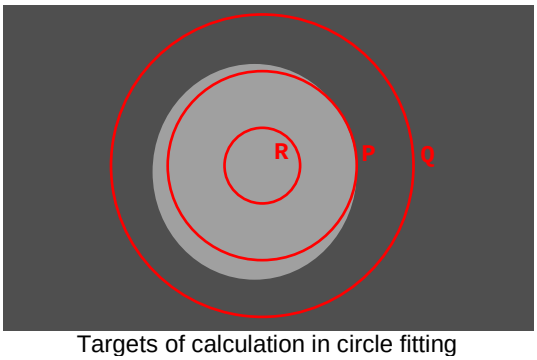
This function performs circle fitting processing of the image at the address specified by src, and outputs the coordinates of the center, the radius, and the score for the circle that was found to the range from the address specified by dst.

In the case of the image in which a single edge can be recognized, the image has different brightnesses at point A in the image and at another point B which is across the edge from point A.



An image having an edge of oblique line

Circle fitting processing starts with the assumptions that a circle is to be found by the search, of a circle P, a concentric circle Q having a larger radius, and a concentric circle R having a smaller radius. The absolute value of the difference in brightness between the regions of the outlines of circles Q and R is calculated by using the above concept.



Targets of calculation in circle fitting

The points at which the brightness values are sampled for the circle having the center coordinate (x,y) and radius r are the 48 points starting from the point (x+r,y) and distributed around the circumference of the circle at an angular interval of 7.5 degrees. If the values of the coordinates of a sampling point are not integers, the decimal fraction in the value is rounded up or down.

The score in circle fitting for a circle having center coordinate (x,y) and radius r is calculated in the way described below.

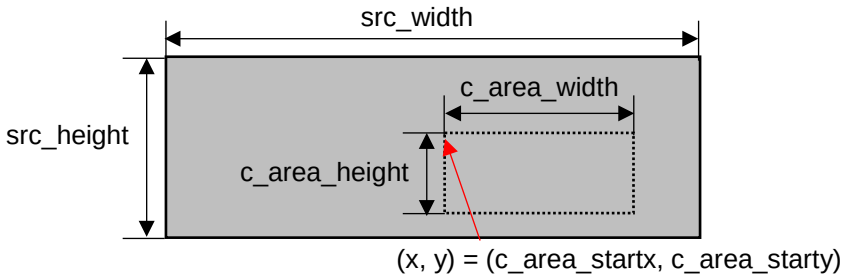
$$score = |(Total\ of\ brightness\ values\ of\ 48\ points\ on\ the\ circumference\ with\ the\ center\ coordinate\ (x,y) and\ radius\ (r\ +\ step)) - Total\ of\ brightness\ values\ of\ 48\ points\ on\ the\ circumference\ with\ the\ center\ coordinate\ (x,y)\ and\ radius\ (r\ -\ step)|$$

The center coordinate and radius are varied to search for the values that deliver the highest score, and the x and y coordinates of the center, radius r, and score of the final result are output.

If scores for multiple coordinates and radii are equal highest, the order of priority listed below is applied to obtain the final result.

1. The smallest radius
2. The smallest y-coordinate value
3. The smallest x-coordinate value

The parameters c_area_startx, c_area_starty, c_area_width, and c_area_height determine the area to be searched for the center of a circle as shown in the figure below. Set the area to be searched for the center of a circle to be wholely within the area of the input image.
The circle fitting processing is performed from the center coordinates (c_area_startx + step * n, c_area_starty + step * n) [n is an integer not less than 0]. However, if part of a circle is outside the area of the input image, it is deemed not to be a circle.

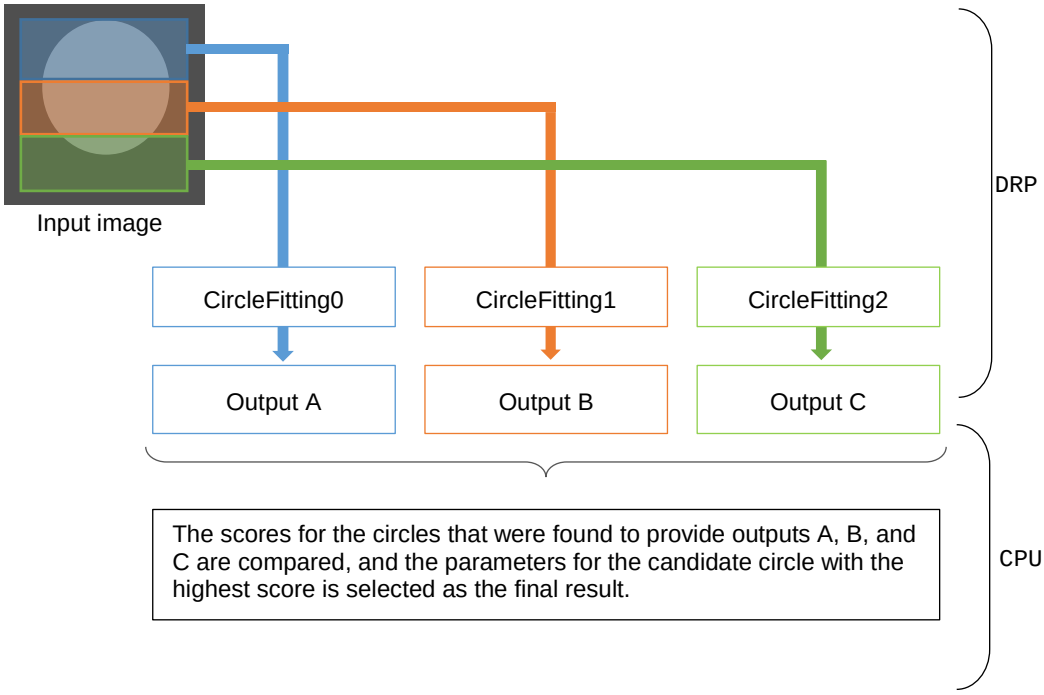


This function allows processing to be segmented with the aid of the CPU.

An example of segmentation for three parallel flows of processing is shown below.

The search area is segmented into the three areas CircleFitting0, CircleFitting1, and CircleFitting2, and the prescribed dst, work, c_area_startx, c_area_starty, c_area_width, and c_area_height are specified for the three respective areas to perform the circle fitting processing from the same center coordinates as those before the segmentation. Use the same settings of src, src_width, src_height, min_radius, max_radius, and step.

After the DRP completes the circle fitting processing, the scores (of the circles that were found) are output to the dst area from CircleFitting0, CircleFitting1, and CircleFitting2, and the highest score is selected as the final result. Segmented processing can thus be realized.



Note	If the parameter settings are such that part or the whole of any candidate circle is out of the area of input image, regardless of the point in the search area that is set as the center, the values of all output variables will always be 0.

4.5.5 FindContours

FindContours

Detects contours in the image and calculate its bounding rectangle

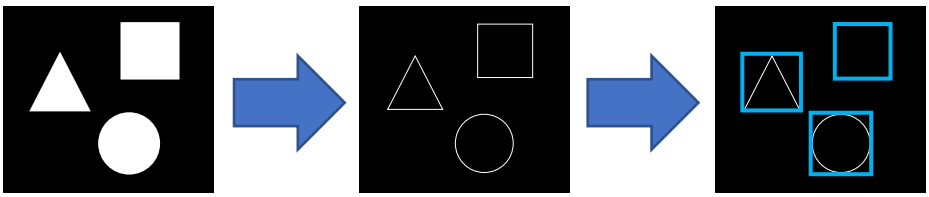
Configuration data file			r_drp_find_contours.dat
Supported version			0.90
Configuration data size (byte)			206816
Header file			r_drp_find_contours.h
Parameter	Structure name		
	r_drp_find_contours_t		
	Member name	Type	Description
	src	uint32_t	Input image address
	dst_rect	uint32_t	Output data (rectangle information) address
	dst_region	uint32_t	Output data (region information) address
	width	uint16_t	Input image width (pixels)
	height	uint16_t	Input image height (pixels)
	work	uint32_t	Work area address
	dst_rect_size	uint32_t	Maximum output number of Rectangle Information (0 to 20,000) (When set to 0, no output)
	dst_region_size	uint32_t	Maximum output number of Region Information (0 to 500,000) Specify the upper limit of the total of region information to be output, not the number per contour (When set to 0, no output)
	threshold_width	uint16_t	Width threshold of rectangle to be detected (1 to width)
	threshold_height	uint16_t	Height threshold of rectangle to be detected (1 to height)
I/O details	Input image	Address: Width (pixels): Height (pixels): Format: Data size:	Specified by src. Specified by width. (16 to 1280, integer multiple of 8) Specified by height. (8 to 960) Binary image (1 byte per pixel), or 8-bits grayscale (1 byte per pixel) Refer to the description for details. (width) × (height) × 1 byte
	Output data (Rectangle Information)	Address: Format: End Data: Data size:	Specified by dst_rect (Specify an address that differs from src.) From the top address, specifications are made in the following order. Upper-left corner x coordinate of Bounding rectangle (2 bytes) Upper-left corner y coordinate of Bounding rectangle (2 bytes) Bounding rectangle width (2 bytes) Bounding rectangle height (2 bytes) Count of region information (4 bytes) Start address of region information (4 bytes) Refer to the description for details. End of rectangle information. All fields are 0 (16 bytes) Refer to the description for details. (Count of rectangles detected + 1) × 16 bytes "+ 1" means the size of End Data The maximum size is (dst_rect_size) × 16 bytes
	Output data (Region Information)	Address: Format: Data size:	Specified by dst_region (Specify an address that differs from src.) From the top address, specifications are made in the following order. x coordinate of one pixel constituting the contour (2 bytes) y coordinate of one pixel constituting the contour (2 bytes) Refer to the description for details. (The total of pixels constituting every contour) × 4 bytes The maximum size is (dst_region_size) × 4 bytes

	Work area	Address: Data size: Description The area used to store data during findcontours processing.	Specified by work. (width) × (height) × 1 byte
Number of tiles	2		
Segmented processing	Not supported.		

Description

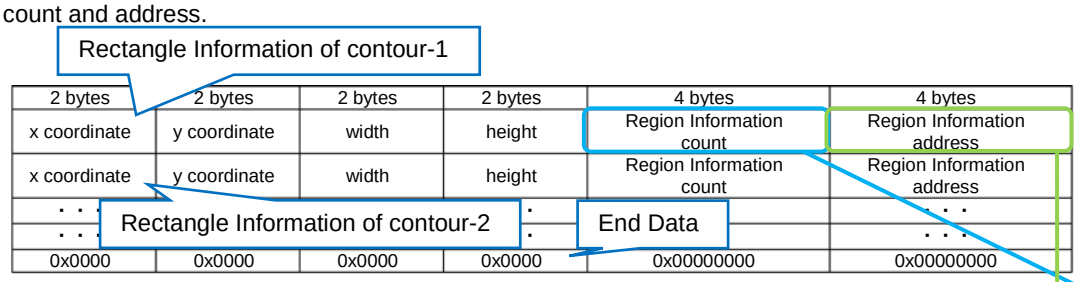
This function performs findcontours processing of the image at the address specified by src and outputs the bounding rectangle information of detected contours to the address specified by dst_rect, and the pixel coordinate constituting detected contours to the address specified by dst_region.

When input the left image below, this function detects three contours as middle image. Then, this function outputs the coordinates of the pixels constituting the contour as "Region Information", and the bounding rectangle is calculated for each detected contour like right image as "Rectangle Information".

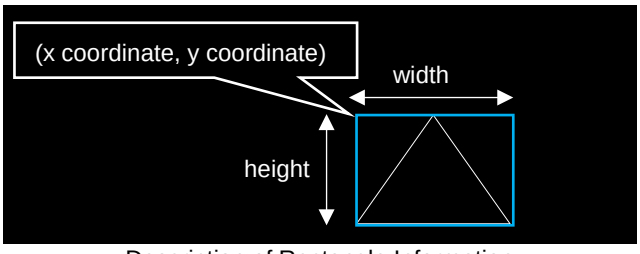


A simplified representation of detection of rectangle in the input image

"Rectangle Information"
This function outputs the bounding rectangle information of detected contours in input image and the corresponding Region Information address and count as shown below. After outputs this data set for the number of detected contours, this function outputs the End Data means end of data.
Keep reading the Rectangle Information until you find the End Data to obtain all the Rectangle Information. Also, you can obtain the corresponding Region Information using the Region Information count and address.

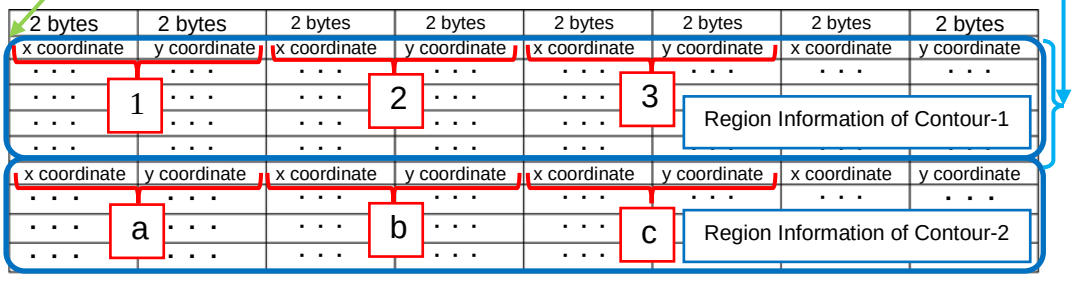


Rectangle Information

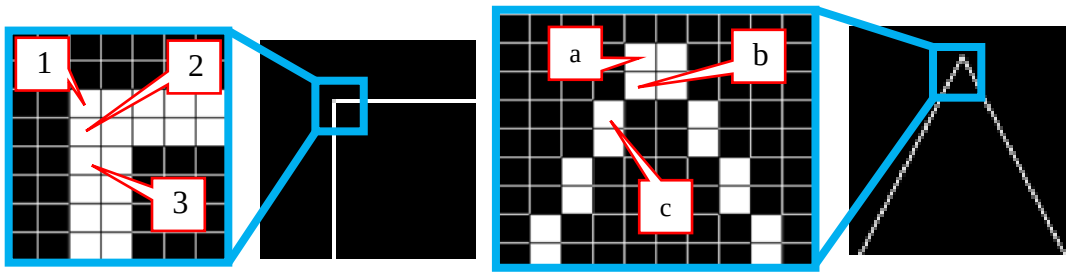


Description of Rectangle Information

"Region Information"
This function outputs (x, y) coordinate of all pixels constituting contours every contour as shown below. This coordinate is expressed in the coordinate system has upper-left corner pixel is (0, 0) in input image.



Region Information



Contour-1 Contour-2

If output data count reaches the value set in dst_rect_size or dst_region_size, the data output of the one that reached the upper limit stops, but the other data output doesn't stop. Moreover, if Both data count reaches upper limit, both data output stop, then DRP terminates.
If the Rectangle Information output count reaches upper limit before output the End Data, The End Data is not output.

This function supposes binary image as input image format.
When input 8-bit grayscale, this function treats pixels with a pixel value of 1 or more as pixels with a pixel value of 1.

When a rectangle width or height is shorter than the parameter threshold_high or threshold_low, this function exclude its Rectangle Information and Region Information from output.

The processing performs by the function is equivalent to that of the OpenCV cv::findcontours function with the specifying of CV_RETR_LIST for mode and CV_CHAIN_APPROX_NONE for method. However, this function output is unique format.
Reference URL: https://opencv.org/

This function allows the same address to be specified for both src and work. However, input image is broken because this function writes out data at the area specified by work during processing.

Note This function output size is dependent on input image.Allocates the sufficient memory area to dst_rect and dst_region to avoid the memory broken by referring Rectangle Information and Region Information of I/O details and setting appropriate values to dst_rect_size and dst_region_size.

4.5.6 MinutiaeExtract

MinutiaeExtract

Extracts minutiae points of fingerprint ridge lines used in fingerprint recognition

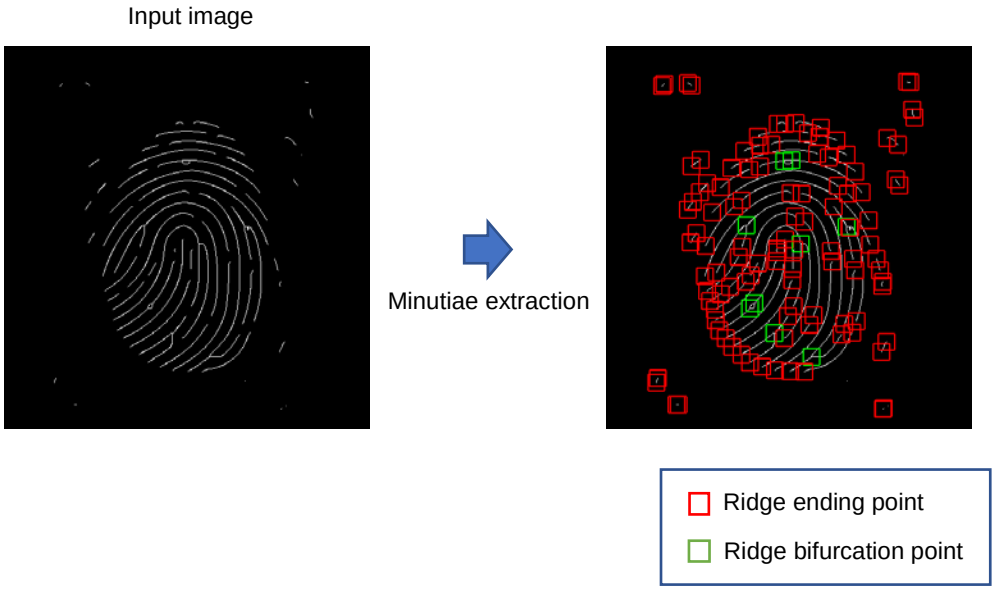
Configuration data file			r_drp_minutiae_extract.dat	
Supported version			0.90	
Configuration data size (byte)			135424	
Header file			r_drp_minutiae_extract.h	
Parameter	Structure name			
	r_drp_minutiae_extract_t			
	Member name	Type	Description	
	src	uint32_t	Input image address	
	width	uint16_t	Image width (pixels)	
	height	uint16_t	Image height (pixels)	
	threshold	uint8_t	Binarization threshold (0 to 255)	
	minutiae_data	uint32_t	Address of minutiae point data	
	minutiae_num	uint32_t	Address of minutiae point count	
	minutiae_max	uint32_t	Max. number of minutiae points (1 to 2048)	
	e_area_startx	uint16_t	X coordinate of extraction area start position	
	e_area_starty	uint16_t	Y coordinate of extraction area start position	
	e_area_width	uint16_t	Width of extraction area	
	e_area_height	uint16_t	Height of extraction area	
I/O details	Input image	Address:	Specified by src.	
		Width (pixels):	Specified by width. (40 to 1280, integer multiple of 4)	
		Height (pixels):	Specified by height. (18 to 960)	
		Format:	8-bit grayscale (1 byte per pixel) Values equal to or less than threshold are treated as black, and other values as white.	
		Data size:	(width) × (height) × 1 byte	
	Extraction area (input)	Start position X coordinate:	Specified by e_area_startx. (4 to width − 36, multiple of 4)	
		Start position Y coordinate:	Specified by e_area_starty. (1 to height − 17)	
		Width (pixels):	Specified by e_area_width. (32 to width − e_area_startx − 4, multiple of 4)	
		Height (pixels):	Specified by e_area_height. (16 to height − e_area_starty − 1)	
		Description This is the area of the input image from which minutiae points are extracted. The 4 pixels on the left and right of the input image cannot be specified as part of the extraction area. The 1 pixel at the top and bottom of the input image cannot be specified as part of the extraction area. Refer to the description for details.		

	Minutiae point data (output)	Address:	Specified by minutiae_data.
		Data size:	minutiae_max × 8 bytes (32 to e_area_width × e_area_height × 8)
		Format:	Starting from the address, X coordinate of extracted minutiae point (2 bytes), Y coordinate of extracted minutiae point (2 bytes), type of extracted minutiae point (1 byte), direction of extracted minutiae point (1 byte), 0 padding (2 bytes)
		Description	
		Starting from the address, the X and Y coordinates, type, and direction of each minutiae point are output as an 8-byte unit. Note that when the number of extracted minutiae points exceeds minutiae_max, only the number of minutiae points specified by minutiae_max is output. The minutiae point type is either ridge ending (0) or ridge bifurcation (1). The minutiae point direction consists of 8 bits of data, starting with bit 0 to the upper left of the target pixel and proceeding clockwise through the adjacent pixels, indicating the positions that are white. Refer to the description for details.	
	Minutiae point count (output)	Address:	Specified by minutiae_num.
		Data size:	4 bytes
		Format:	Number of extracted minutiae points (4 bytes)
		Description	
		The number of extracted minutiae points is output. The actual number of minutiae points extracted is output, even if it exceeds minutiae_max. Refer to the description for details.	
Number of tiles	3		
Segmented processing	Not supported. However, segmented processing can be achieved in combination with processing on the CPU. Refer to the description for details.		

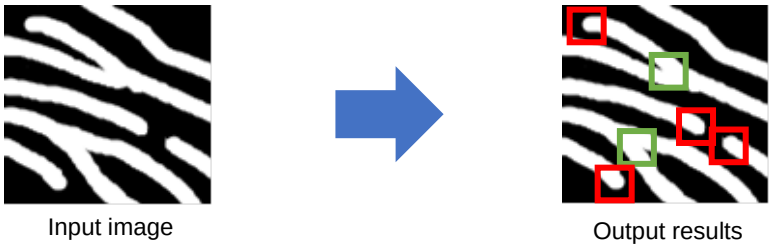
Description

This function binarizes the image at the address specified by src and extracts minutiae points to identify ridge ending and bifurcation points in the image. It is necessary that the input image undergo thinning in order to achieve accurate minutiae detection. The extraction results are output as minutiae_data, and the number of minutiae points extracted is output as minutiae_num.

This function performs processing up to the extraction of minutiae points. The extracted minutiae point data typically requires further processing, such as the deletion of unnecessary minutiae points. The MinutiaeDelete function contained in the library is one example of such processing. For details, refer to the description of the MinutiaeDelete function.

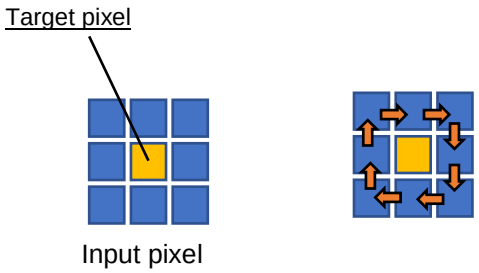


In minutiae extraction, fingerprint minutiae points are identified according to information on target pixels and their adjacent pixels in an image that has been subjected to thinning. The data on each minutiae point includes the X and Y coordinates within the fingerprint, whether the point is a ridge ending or bifurcation point, and the direction of the ending or bifurcation.

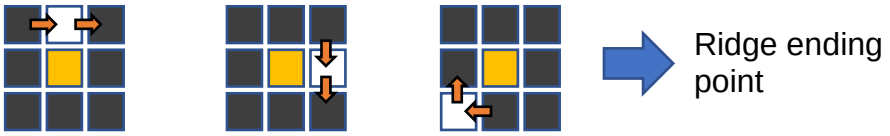


In minutiae extraction, a range encompassing the single pixels adjacent to the target pixel (range of 3 × 3 pixels) is examined, and the number of times the color changes between one pixel and the one next to it (white to black, or black to white) are counted. If the number of changes is two, the point is classified as a ridge ending point, and if it is five or more, the point is classified as a ridge bifurcation point.

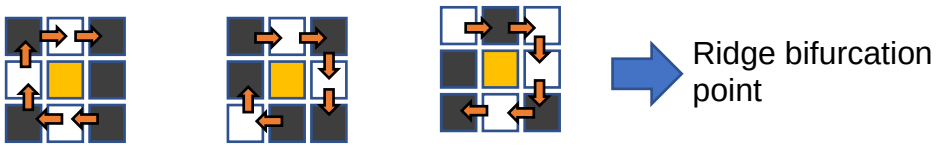
The value of threshold is used to determine whether a pixel is white or black. If the brightness of the pixel in the input image is greater than threshold, it is considered to be white.



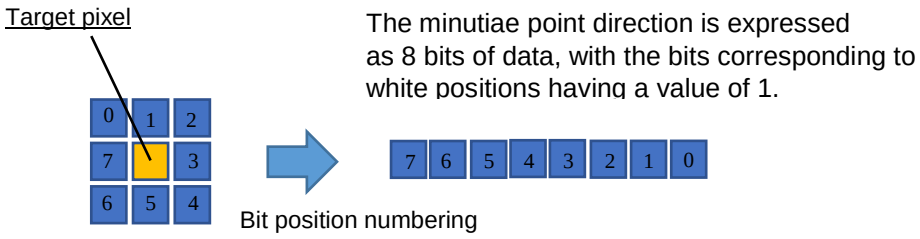
- Examples with two changes



- Examples with five or more changes



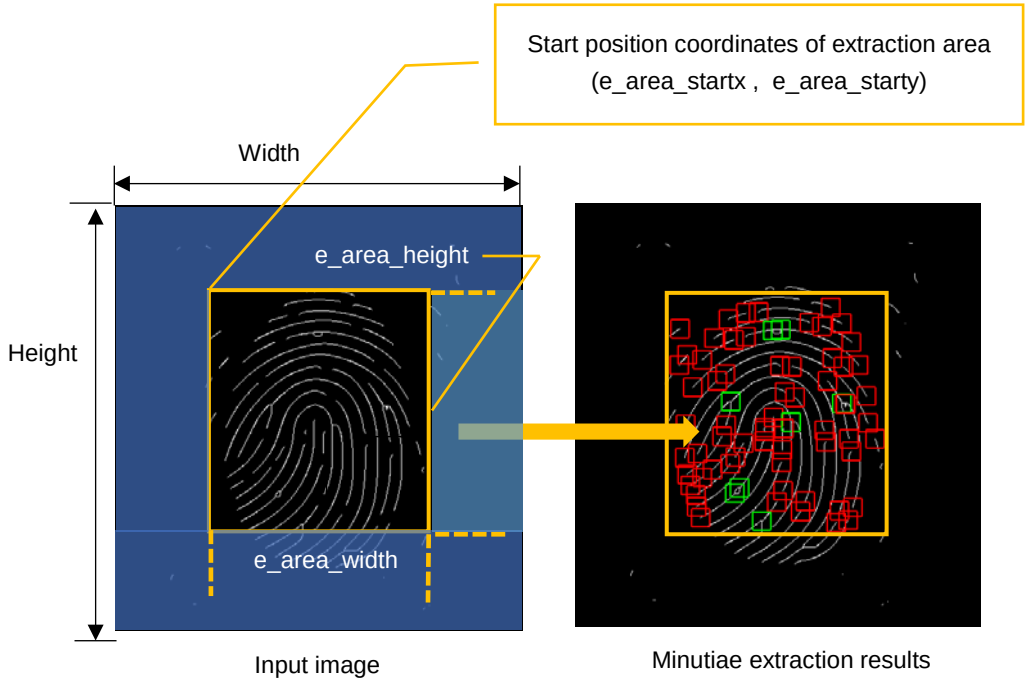
In minutiae extraction, direction information is extracted in addition to bifurcation and ending point information. The minutiae point direction information indicates the positions adjacent to the target pixel that are white, converted into 8 bits of data.



- Example of ending point and direction

- Example of bifurcation point and direction

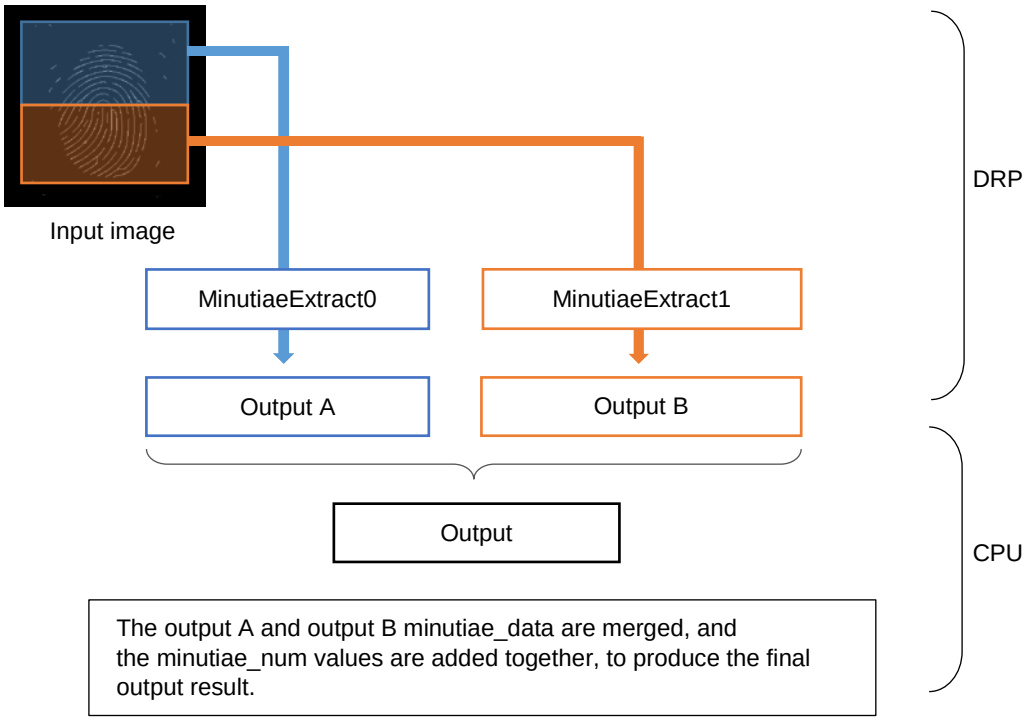
In minutiae extraction, it is possible to specify the area of the image at the address specified by src that is actually subject to processing. This area can be specified by setting the extraction area width in e_area_width, height in e_area_height, and start position coordinates (e_area_startx and e_area_starty).



With this function, segmented processing can be achieved by using the CPU.
In the following example, two areas are processed in parallel.

Divide the search area into two areas, MinutiaeExtract0 and MinutiaeExtract1, and specify specific minutiae_data, minutiae_num, minutiae_max, e_area_startx, e_area_starty, e_area_width, and e_area_height values for each, as when performing minutiae extraction without segmentation. For each segment, set the same values for src, width, height, and threshold.

When minutiae extraction completes on the DRP, merge the output minutiae point data in the minutiae_data areas of MinutiaeExtract0 and MinutiaeExtract1, and add the minutiae point counts (minutiae_num) together to implement segmented processing.



Note	None

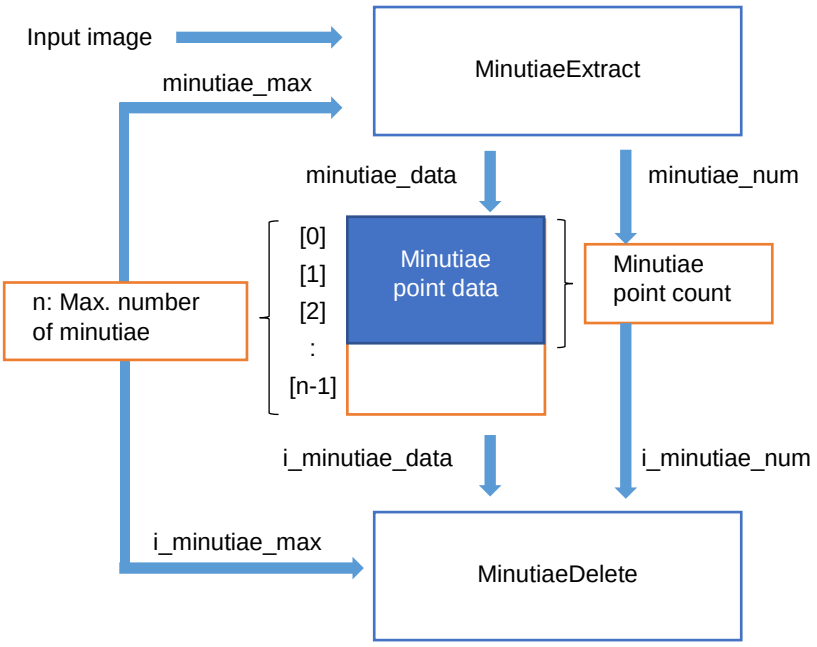
4.5.7 MinutiaeDelete

MinutiaeDelete

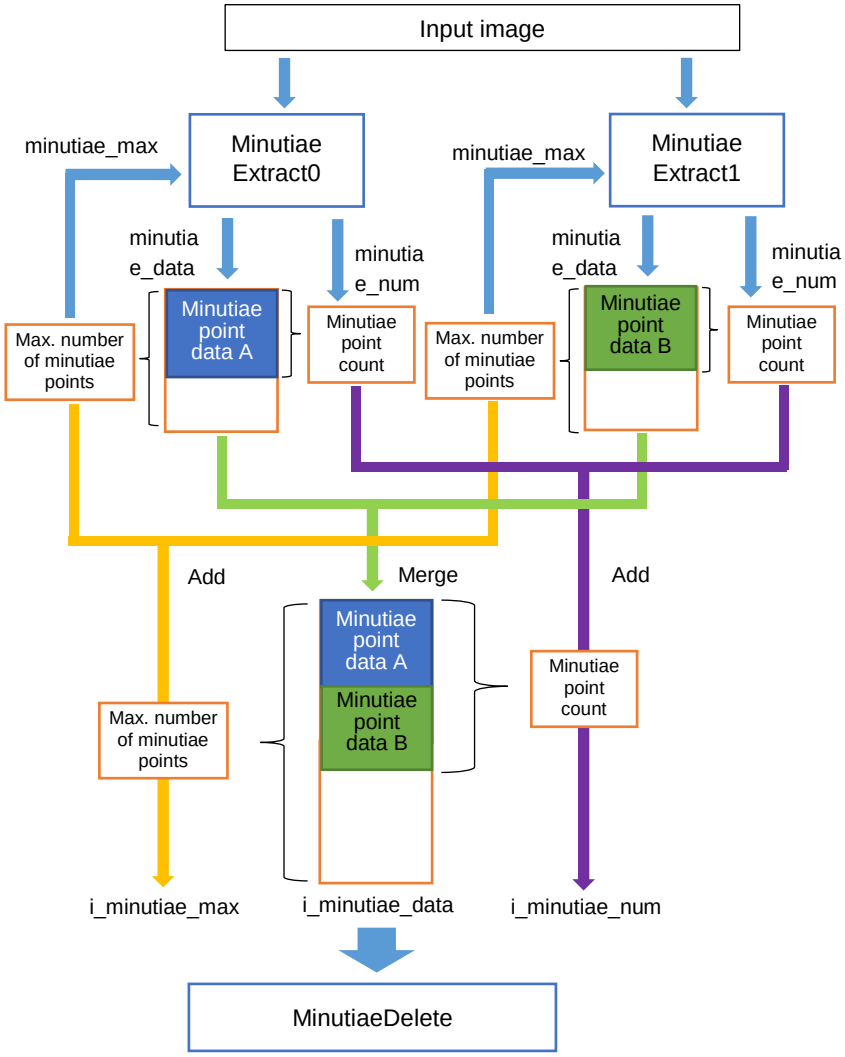
Deletes minutiae points of fingerprint ridge lines used in fingerprint recognition

Configuration data file		r_drp_minutiae_delete.dat	
Supported version		0.90	
Configuration data size (byte)		113024	
Header file		r_drp_minutiae_delete.h	
Parameter	Structure name		
	r_drp_minutiae_delete_t		
	Member name	Type	Description
	trust_map	uint32_t	Address of trustworthiness information
	width	uint16_t	Width of trustworthiness information (pixels)
	height	uint16_t	Height of trustworthiness information (pixels)
	i_minutiae_data	uint32_t	Address of input minutiae point data
	i_minutiae_num	uint32_t	Address of input minutiae point count
	i_minutiae_max	uint32_t	Max. input minutiae point count (1 to 2048)
	o_minutiae_data	uint32_t	Address of output minutiae point data
	o_minutiae_num	uint32_t	Address of output minutiae point count
	work	uint32_t	Address work area
	First deletion		
	del1_distance	uint16_t	First deletion distance specification (0 to 65535)
	del1_probability	uint8_t	First deletion trustworthiness information specification (0 to 255)
	del1_bifurcation	uint8_t	First deletion bifurcation point deletion suppression specification (0 to 255)
	Second deletion		
	del2_distance	uint16_t	Second deletion distance specification (0 to 65535)
	del2_count	uint8_t	Second deletion minutiae point count specification (0 to 255)
	del2_bifurcation	uint8_t	Second deletion bifurcation point deletion suppression specification (0 to 255)
	Third deletion		
	del3_distance_s	uint16_t	Third deletion distance specification (same type) (0 to 65535)
	del3_distance_d	uint16_t	Third deletion distance specification (different type) (0 to 65535)
	del3_probability	uint8_t	Third deletion trustworthiness information specification (0 to 255)
	del3_bifurcation	uint8_t	Third deletion bifurcation point deletion suppression specification (0 to 255)

I/O details	Input minutiae point count	Address:	Specified by i_minutiae_num.
		Data size:	4 bytes
		Format:	Number of minutiae points to be deleted (4 bytes)
		Description	
		Inputs the number of minutiae points to be deleted.	
		However, if the value specified for i_minutiae_num exceeds i_minutiae_max, only the number of minutiae points specified by i_minutiae_max is deleted.	
	Input minutiae point data	Address:	Specified by i_minutiae_data.
		Data size:	Input minutiae point count × 8 bytes
		Format:	Starting from the address, X coordinate of extracted minutiae point (2 bytes), Y coordinate of extracted minutiae point (2 bytes), type of extracted minutiae point (1 byte), direction of extracted minutiae point (1 byte), 0 padding (2 bytes)
		Description	
		Starting from the address, the X and Y coordinates, type, and direction of each minutiae point are input as an 8-byte unit, up to the specified input minutiae point count. When inputting MinutiaeExtract results, specify the address of minutiae_data in i_minutiae_data and the address of minutiae_num in i_minutiae_num, and set the value of miutiae_max to i_minutiae_max.	
		[Segmented processing not used]	



[Segmented processing used]



If MinutiaeExtract was processed in segments, store the merged minutiae point data at the addresses specified by i_minutiae_data and i_minutiae_num.

The minutiae point type is either ridge ending (0) or ridge bifurcation (1).

The minutiae point direction consists of 8 bits of data, starting with bit 0 to the upper left of the target pixel and proceeding clockwise through the adjacent pixels, indicating the positions that are white.

Refer to the description of MinutiaeExtract for details.

Trustworthiness information (input)	Address:	Specified by trust_map.
	Width (pixels):	Specified by width. (40 to 1280, integer multiple of 4)
	Height (pixels):	Specified by height. (18 to 960)
	Format:	8 bits (1 byte per pixel)
	Data size:	(width) × (height) × 1 byte

Description

This information is used to determine whether or not minutiae points are trustworthy. A higher numeric value corresponds to a higher level of trustworthiness.

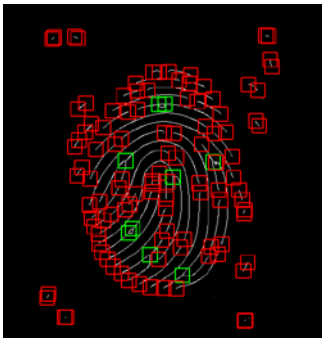
Refer to the description for details.

	Output minutiae point count	Address:	Specified by o_minutiae_num.
		Data size:	4 bytes
		Format:	Number of minutiae points after deletion (4 bytes)
		Description This specifies the number of minutiae points to be output after deletion.	
	Output minutiae point data	Address:	Specified by o_minutiae_data.
		Data size:	i_minutiae_max × 8 bytes
		Format:	Starting from the address, X coordinate of extracted minutiae point (2 bytes), Y coordinate of extracted minutiae point (2 bytes), type of extracted minutiae point (1 byte), direction of extracted minutiae point (1 byte), 0 padding (2 bytes)
		Description Starting from the address, the X and Y coordinates, type, and direction of each minutiae point are output after deletion as an 8-byte unit, up to the specified output minutiae point count. The maximum value of the output minutiae point count is i_minutiae_max, so the data size only needs to accommodate i_minutiae_max. The minutiae point type is either ridge ending (0) or ridge bifurcation (1). The minutiae point direction consists of 8 bits of data, starting with bit 0 to the upper left of the target pixel and proceeding clockwise through the adjacent pixels, indicating the positions that are white. Refer to the description of MinutiaeExtract for details.	
	Work area	Address:	Specified by work.
		Data size:	i_minutiae_max × 16 bytes
		Description This is where data is stored while processing of minutiae deletion is in progress.	
Number of tiles	2		
Segmented processing	Not supported		

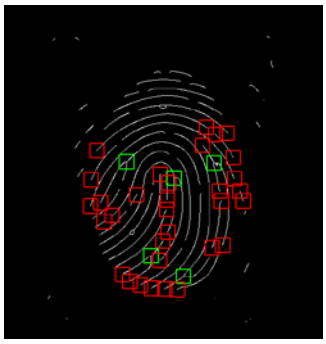
Description

This function performs first deletion, second deletion, and third deletion, based on the minutiae point information (specified by i_minutiae_data and i_minutiae_num) extracted by MinutiaeExtract and the trustworthiness information, then outputs minutiae point information to o_minutiae_data and o_minutiae_num.
Deletion is not performed if the minutiae point count is eight or less. The first deletion, second deletion, and third deletion portions of the process are each described separately below.

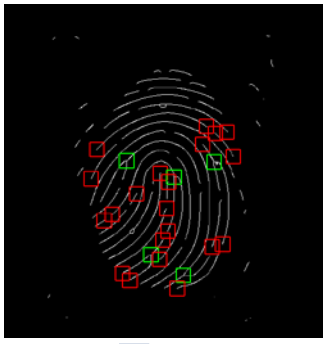
Minutiae extraction results



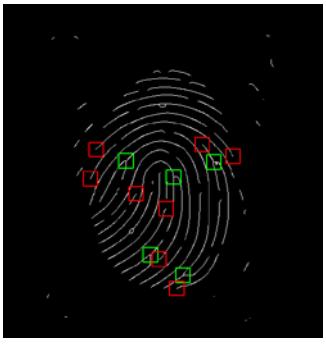
First deletion

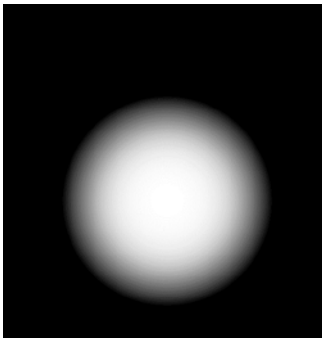


Second deletion



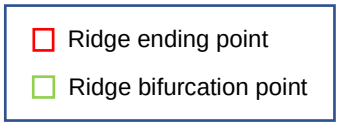
Third deletion





[Trustworthiness information]

Locations where the brightness is low are judged to be lower in trustworthiness and not considered to be valid minutiae points.

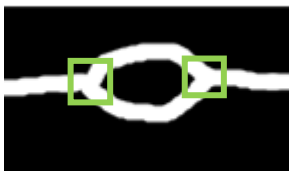


[First deletion]

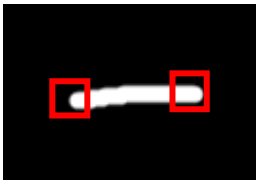
In the first deletion, minutiae points that meet either of the following two conditions are delated:

- Minutiae points of the same type that are at the specified distance, as viewed from the target minutiae point
- Minutiae points with low trustworthiness

These conditions are used because minutiae points of the same type may not be true minutiae points if they result from a hole in a line or a short line.



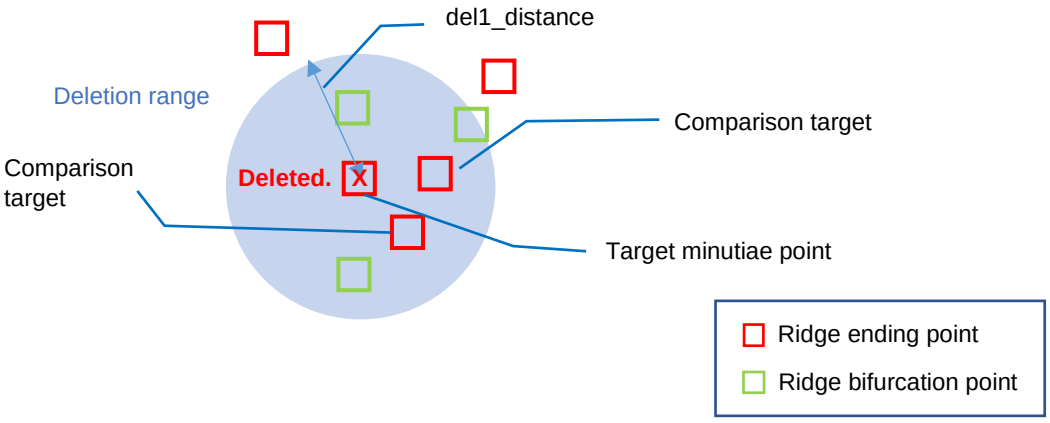
Hole in line



Short line

In the first deletion, the target minutiae point (coordinates XA,YA) is compared to all other minutiae points (coordinates XB,YB), and a determination to delete the minutiae point is made when the type is the same and the relationship with del1_distance meets the following condition:

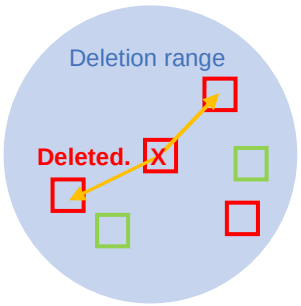
$$(XA - XB)^2 + (YA - YB)^2 < del1\_distance$$



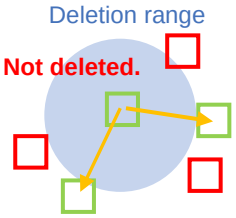
The deletion processing is able to avoid deleting bifurcation points in the first deletion.
When deleting based on distance, a determination to delete is made when the bifurcation point distance is twice del1_bifurcation.

This means that bifurcation points are deleted when they meet the following condition:

$$\{(XA - XB)^2 + (YA - YB)^2 \} \times del1\_bifurcation < del1\_distance$$



If center pixel is ending point

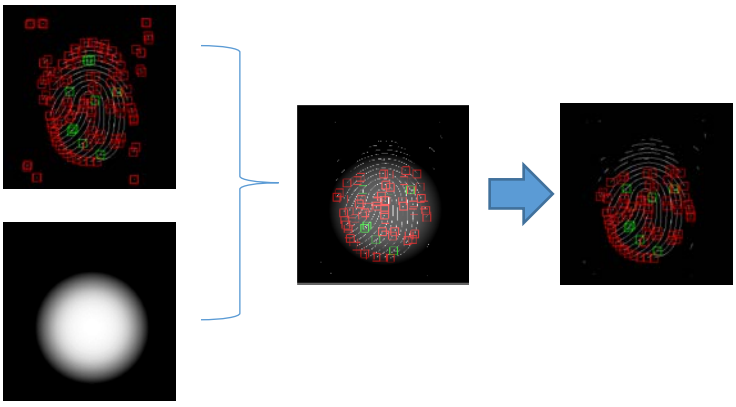


If center pixel is bifurcation point

Bifurcation points for which the radius is 1 / [del1_bifurcation] are not deleted.

When deleting based on low trustworthiness, the target minutiae point is judged to have low trustworthiness and is deleted when the trustworthiness information value is less than del1_probability.

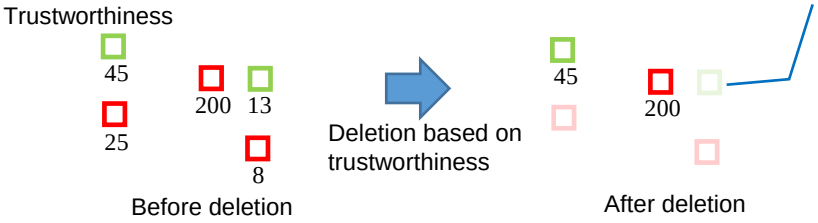
Note that the processing avoids deleting bifurcation points when making deletions based on low trustworthiness. Bifurcation points are deleted when (trustworthiness information value × del1_bifurcation) is less than del1_probability.



Trustworthiness

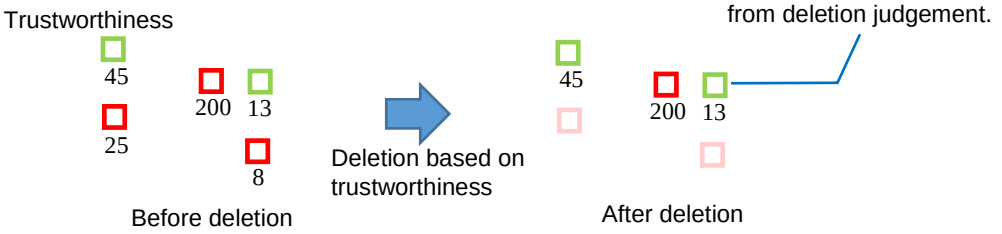
If del1_probability = 30 and del1_bifurcation = 1

Judgement is made to delete minutiae points with value of less than 30.



If del1_probability = 30 and del1_bifurcation = 3

13 × 3 = 39, so exempted from deletion judgement.



To not make deletions based on distance, set del1_distance to 0. To not make deletions based on trustworthiness, set del1_probability to 0.
When both del1_distance and del1_probability are set to 0, first deletion processing does not take place.

[Second deletion]

In the second deletion, minutiae points that meet the following condition are deleted:

- Minutiae points that are at the specified distance, as viewed from the target minutiae point, and exceed the specified number.

This condition is used because when minutiae points are clustered together, they may be the result of noise.

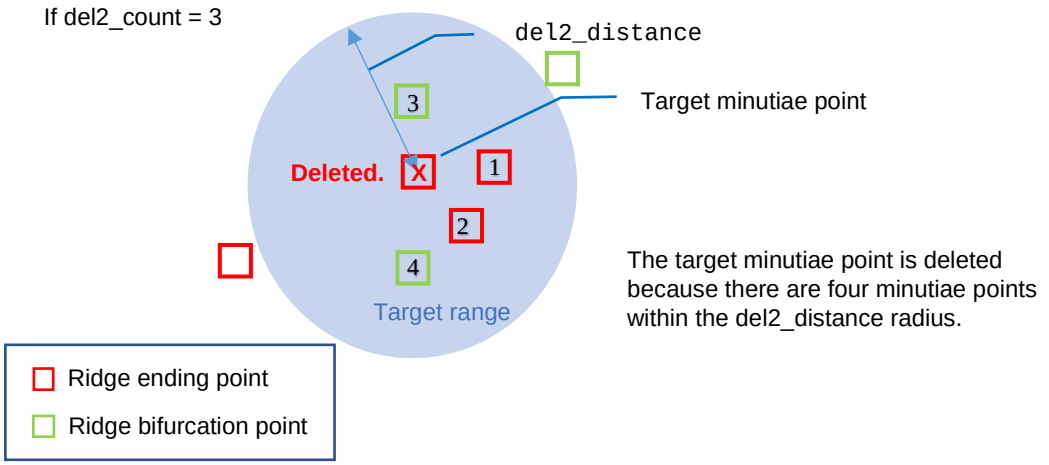
In the second deletion, the target minutiae point (coordinates XA,YA) is compared to all other minutiae points (coordinates XB,YB), and a determination to delete the minutiae point is made when it is a ridge ending point and the result of the formula below is del2_count or greater.

$$(XA – XB)^2 + (YA – YB)^2 < del2\_distance$$

When the target minutiae point is a bifurcation point, and the number of minutiae points that satisfy the formula below is del2_count or greater, the target minutiae point is deleted.

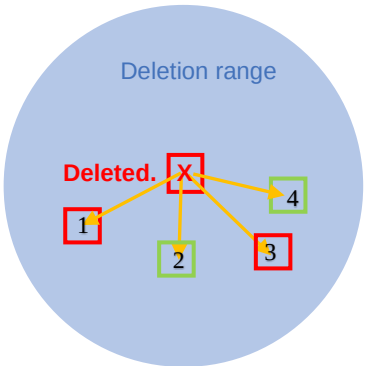
$$\{(XA – XB)^2 + (YA – YB)^2 \} \times del2\_bifurcation < del2\_distance$$

If del2_count = 3

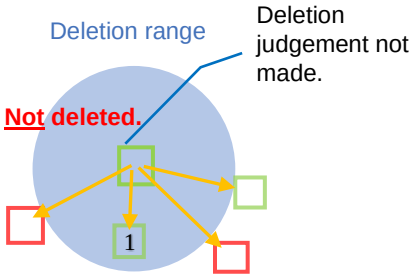


The deletion processing is able to avoid deleting bifurcation points in the second deletion.

If del2_count = 3



If center pixel is ending point



If center pixel is bifurcation point

Not deleted, because there is only one minutiae point.

To not make deletions based on distance, set del2_distance to 0. When del2_distance is set to 0, second deletion processing does not take place.

[Third deletion]

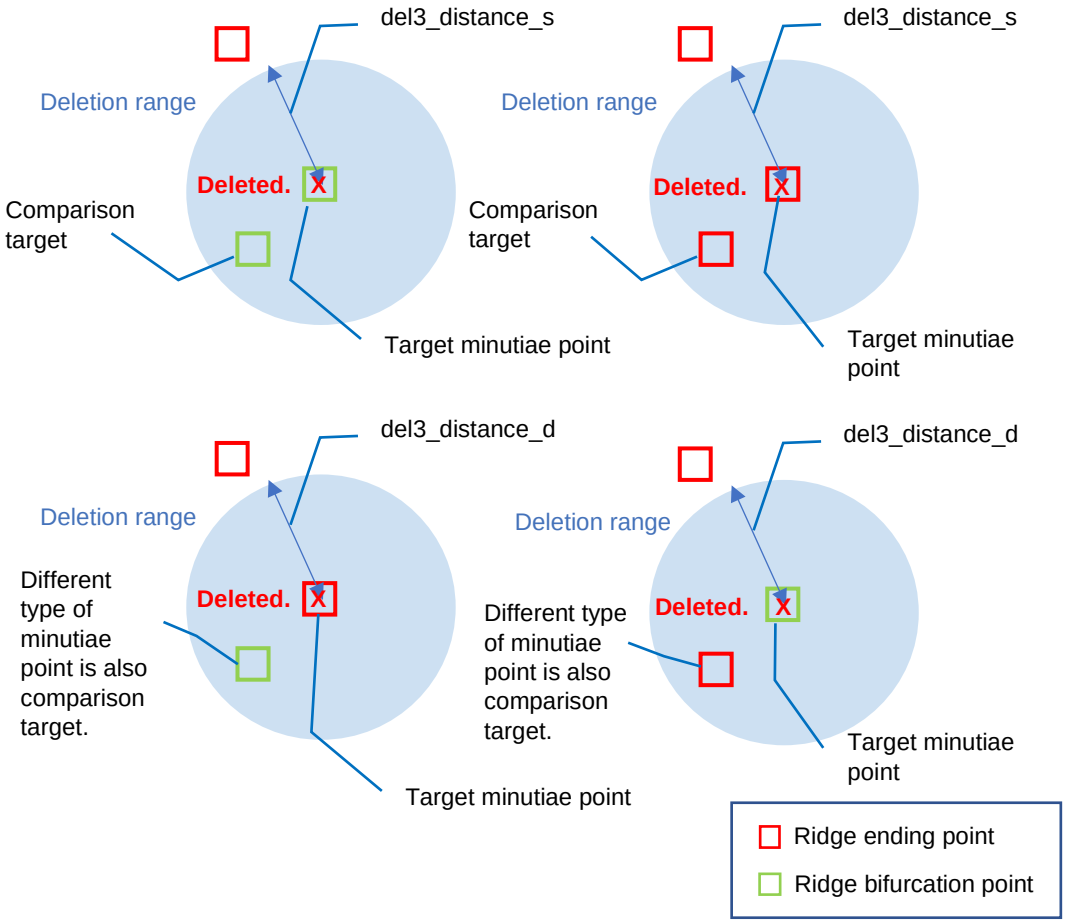
Third deletion processing takes place after first deletion and second deletion processing.
The deletion conditions are the same as those of the first deletion, but deletion is performed on minutiae points (including points of different types) that are near each other. When the target minutiae point (coordinates XA,YA) is a bifurcation point, deletion suppression based on del3_bifurcation applies. The comparison target minutiae points have (coordinates XB,YB).

When the minutiae points are of the same type, deletion occurs when del3_distance_s satisfies the formula below. (When the target pixel is an ending point, del3_bifurcation = 1.)

$$\{(XA - XB)^2 + (YA - YB)^2 \} \times del3\_bifurcation < del3\_distance\_s$$

When the minutiae points are of different types, deletion occurs when del3_distance_d satisfies the formula below. (When the target pixel is an ending point, del3_bifurcation = 1.)

$$\{(XA - XB)^2 + (YA - YB)^2 \} \times del3\_bifurcation < del3\_distance\_d$$



As with the first deletion, the third deletion also performs deletions when trustworthiness is low. This processing uses del3_probability and del3_bifurcation, and the conditions are the same as those of the first deletion.

To not make deletions based on distance, set del3_distance_s and del3_distance_d to 0. To not make deletions based on trustworthiness, set del3_probability to 0.
When del3_distance_s, del3_distance_d, and del3_probability are all set to 0, third deletion processing does not take place.

	Setting not to perform all deletions of first deletion, second deletion and third deletion is prohibited.
Note	None

4.5.8 Thinning

Thinning

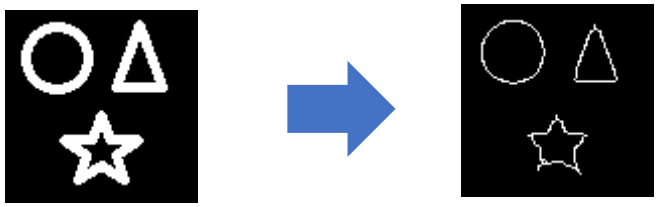
Outputs an image on which thinning has been performed

Configuration data file			r_drp_thinning.dat
Supported version			0.90
Configuration data size (byte)			119872
Header file			r_drp_thinning.h
Parameter	Structure name		
	r_drp_thinning_t		
	Member name	Type	Description
	src	uint32_t	Input data address
	dst	uint32_t	Output data address
	width	uint16_t	Image width (pixels)
	height	uint16_t	Image height (pixels)
	result	uint32_t	Address of processing results
	top	uint8_t	1: Top edge border processing 0: No top edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the top edge of the source image.
	bottom	uint8_t	1: Bottom edge border processing 0: No bottom edge border processing Specify 1 if the input image is not segmented. For segmenting the input image for processing, specify 1 if the input image reaches the bottom edge of the source image.
	step	uint8_t	Type of repeat processing 0: step1 1: step2 Specify step1 for odd-numbered processing repetitions. Specify step2 for even-numbered processing repetitions. Refer to the description for details.
	reverse	uint8_t	0: Thinning is performed on white portions. 1: Thinning is performed on black portions.
	threshold	uint8_t	Binarization threshold (0 to 255)
I/O details	Input image	Address:	Specified by src.
		Width (pixels):	Specified by width. (128 to 1280, integer multiple of 8)
		Height (pixels):	Specified by height. (16 to 960)
		Format:	8-bit grayscale (1 byte per pixel) A value of 0 is treated as black, and values of 1 or greater are treated as white.
		Data size:	(width) × (height) × 1 byte
	Output image	Address:	Specified by dst.
		Width (pixels):	Same as input image
		Height (pixels):	Same as input image
		Format:	8-bit grayscale (0 or 255) (1 byte per pixel) Black is output as 0 and white as 255.
		Data size:	(width) × (height) × 1 byte

	Processing results	Address:	Specified by result. (Specify an address that differs from src or dst.)
		Data size:	4 bytes
		Format:	Number of pixels in white portions changed to black (4 bytes) (0 to width × height)
		Description This is the area where the number of pixels in white portions changed to black (or in black portions changed to white) as a result of thinning is stored. When the number of pixels in white portions changed to black (or in black portions changed to white) is 0, it means that thinning has completed. Refer to the description for details.	
Number of tiles	3		
Segmented processing	Supported		

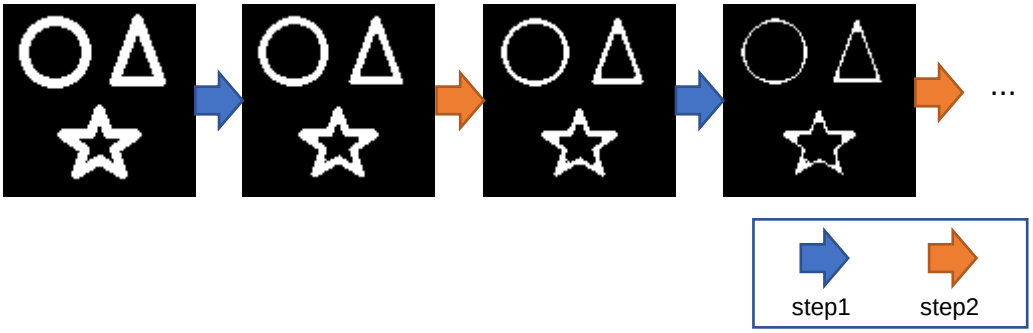
Description

This function binarizes the image at the address specified by src, performs thinning on the white portions (or black portions), and outputs the thinning results to the address specified by dst. It also outputs the number of pixels in white portions changed to black (or in black portions changed to white) to the address specified by result. In the description below it is assumed that reverse is set to 0. For the processing when reverse is set to 1, simply replace the phrase "changing white portions to black" in the description with "changing black portions to white." During binarization, pixels where the input data exceeds threshold are treated as white, and pixels where it is equal to or less than threshold are treated as black.



Input data Output data

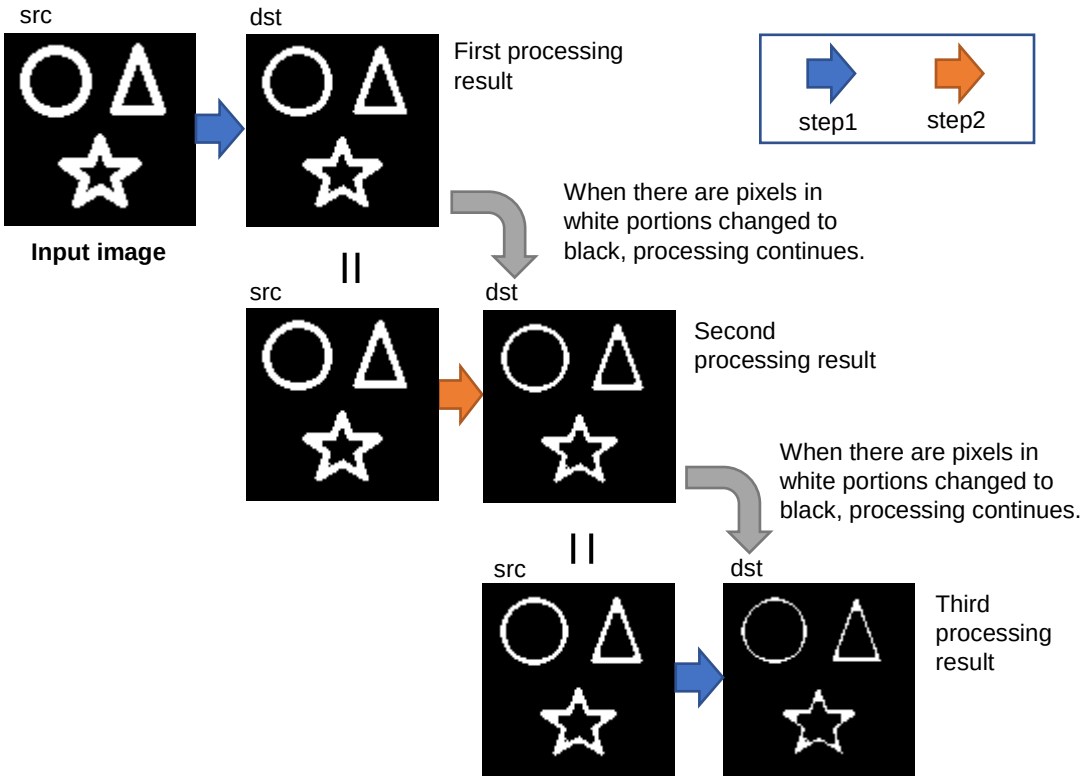
Thinning requires repeat processing based on algorithms for changing white portions to black, and this function uses Zhang-Suen algorithms for this purpose. There are two types of Zhang-Suen algorithm (called step1 and step2 for convenience), and these are applied in alternation. Both step1 and step2 have their own conditions for changing white portions to black, and these conditions are applied to a 3 × 3 grid of pixels with the target pixel in the center. For border processing, this function treats pixels outside the range of the input image as black.



When pixels in white portions are changed to black during step1 or step2 processing (the processing results value specified by result is 1 or greater), the resulting output image is set as the input image, and processing continues, with step1 followed by step2, or step2 followed by step1, as the case may be.
When no white portions are changed to black during step1 or step2 processing (the processing results value specified by result is 0), thinning ends.

When segmented processing is used, repeat processing continues until there are no more pixels in white portions changed to black in any of the processing segments.

It is possible to end thinning while some pixels in white portions changed to black remain in order to fix the maximum processing duration. However, this may produce an incomplete result, on which thinning has not completed, as the output image.



If segmented processing of this function is not used, the same address may be specified for both src and dst.

Note	None

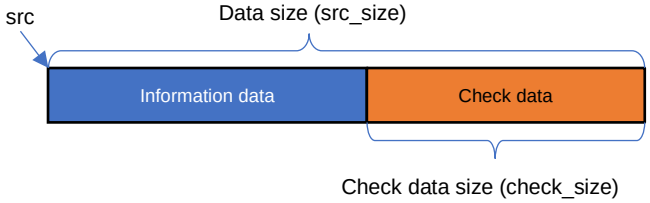
4.6 Other

4.6.1 ReedSolomon

ReedSolomon

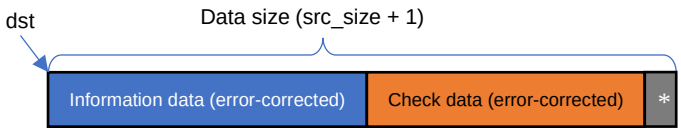
Performs error correction using Reed-Solomon codes (fixed primitive polynomial)

Configuration data file			r_drp_reed_solomon.dat	
Supported version			0.91	
Configuration data size (byte)			118848	
Header file			r_drp_reed_solomon.h	
Parameter	Structure name			
	r_drp_reed_solomon_t			
	Member name	Type	Description	
	src	uint32_t	Input data address	
	dst	uint32_t	Output data address	
	src_size	uint16_t	Input data size (bytes)	
	check_size	uint16_t	Check data size (bytes)	
I/O details	Input data	Address:	Specified by src.	
		Data size:	Specified by src_size. (2 to 254)	
		Information data:	Data for error correction. (1 to 253)	
		Check data:	Check data added during encoding for use in error correction	
		Check data size:	Specified by check_size. (1 to 127)	



The information data and check data are input as Reed-Solomon encoded results. The encoded results are assumed to have the zero-dimensional coefficient of the primitive polynomial as the LSB.

	Output data	Address:	Specified by dst.
		Data size:	src_size + 1 byte. (Error correction)
		Information data (error-corrected):	Error corrected information data. (Data size is same the information data in input data)
		Check data (error-corrected):	Error corrected check data. (Data size is same the check data in input data)
		Error correction:	Data indicating error correction data. (1 byte)



*: Error correction

Number of tiles	1
Segmented processing	Not supported

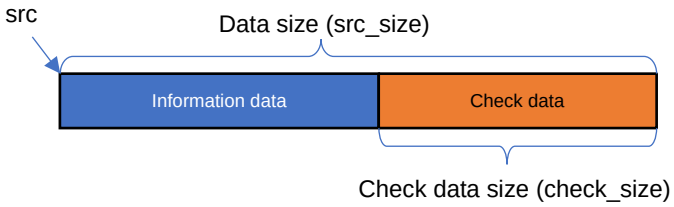
Description	This function performs Reed-Solomon decoding using the specification listed below on the input data at the address specified by src, and outputs error-corrected data and the error correction result to the address specified by dst. To specify a user-defined primitive polynomial, use the ReedSolomonGf8 function. Reed-Solomon decoding specification: • Galois field: GF(2^8) • Primitive polynomial over Galois field: $X^8+X^4+X^3+X^2+1$ • Number of bits per symbol: 8 The result of error correction is stored in "Error correction" appended at the end of output data. "Error correction" is stored "0" if the error correction succeeded, and "1" if failed. The number of symbols that can be used for error correction is equal to floor (check_size ÷ 2). Therefore, no error correction takes place and the output data remains unchanged if check_size is set to 1. In this case 0 is output as the error correction result. This function performs decoding consisting of syndrome calculation, Euclidean algorithm, chain searching, and error value calculation, in that order. If no errors are detected, the processing ends with syndrome calculation. If errors are detected, the processing proceeds through error value calculation. Note that the larger the number of errors, the more processing time is required for the Euclidean algorithm and error value calculation.
Note	If the number of errors in the input data exceeds the number of symbols available for correction, false corrections may result. In some cases, even though false corrections lead to inaccurate decoding and error correction fails, the value of "Error correction" may not indicate failure (1), and symbols that are not in error may be changed.

4.6.2 ReedSolomonGf8

ReedSolomonGf8

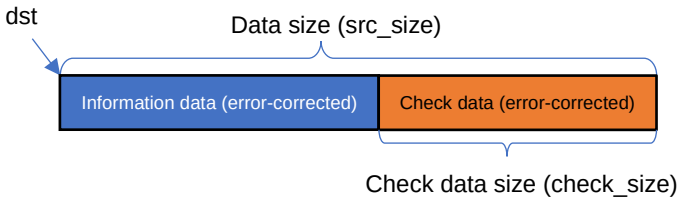
Performs error correction using GF(2^8) Reed-Solomon codes

Configuration data file		r_drp_reed_solomon_gf8.dat	
Supported version		0.91	
Configuration data size (byte)		120352	
Header file		r_drp_reed_solomon_gf8.h	
Parameter	Structure name		
	r_drp_reed_solomon_gf8_t		
	Member name	Type	Description
	src	uint32_t	Input data address
	dst	uint32_t	Output data address
	correct_addr	uint32_t	Address to which error correction count is output
	src_size	uint16_t	Input data size (bytes)
	check_size	uint16_t	Check data size (bytes)
	primitive	uint16_t	Primitive polynomial over Galois field (zero-dimensional coefficient as LSB) 0x11D in case of $\chi^8 + \chi^4 + \chi^3 + \chi^2 + 1$
I/O details	Input data	Address:	Specified by src.
		Data size:	Specified by src_size. (2 to 255 bytes)
		Information data:	Data for use in error correction (1 to 254 bytes)
		Check data:	Check data added during encoding for use in error correction
		Check data size:	Specified by check_size. (1 to 127 bytes)



The information data and check data are input as Reed-Solomon encoded results. The encoded results are assumed to have the zero-dimensional coefficient of the primitive polynomial as the LSB.

For reference, the correspondence between the exponential expression and vector expression of the Galois field is α^0, α^1, α^2 to 0x01, 0x02, 0x04.

	Output data	Address:	Specified by dst.
		Data size:	src_size
		Information data (error-corrected):	Error corrected data. (Size is same as that of information data.)
		Check data (error-corrected):	Check data for error correction. (Size is same as that of check data.)
			
	Error correction count	Address:	Specified by correct_addr.
		Data size:	1 byte
		Description	
		The result of Reed-Solomon decoding is output. The decoding result is the number of errors that were corrected. If there were no errors in the input data, 0 is output. If error correction fails, 0xff is output.	
Number of tiles	1		
Segmented processing	Not supported		

Description	This function performs Reed-Solomon decoding using the specification listed below on the input data at the address specified by src, and outputs error-corrected data to the address specified by dst and the error correction count to the address specified by correct_addr. Reed-Solomon decoding specification: • Galois field: GF(2^8) • Primitive polynomial over Galois field: Specified by primitive • Number of bits per symbol: 8 The zero-dimensional coefficient of the primitive polynomial over Galois field is set as the LSB. For example, primitive is set to 0x11D to specify $\chi^8 + \chi^4 + \chi^3 + \chi^2 + 1$. The number of errors corrected by Reed-Solomon decoding is stored at the address specified by correct_addr. When error correction is successful, a value equal to the correction count is stored at the address specified by correct_addr, and 0xff is stored when error correction fails. When error correction fails, no corrections are applied and the output data remains unchanged. The number of symbols that can be used for error correction is equal to floor (check_size ÷ 2). Therefore, no error correction takes place and the output data remains unchanged if check_size is set to 1. A value of 0xff is output rather than 0 as the error correction count. This function performs decoding consisting of syndrome calculation, Euclidean algorithm, chain searching, and error value calculation, in that order. If no errors are detected, the processing ends with syndrome calculation. If errors are detected, the processing proceeds through error value calculation. Note that the larger the number of errors, the more processing time is required for the Euclidean algorithm and error value calculation.
Note	If the number of errors in the input data exceeds the number of symbols available for correction, false corrections may result. In some cases, even though false corrections lead to inaccurate decoding and error correction fails, the value of "Error correction" may not indicate failure (0xff), and symbols that are not in error may be changed.

4.6.3 Histogram

Histogram

Generates a histogram from the input image

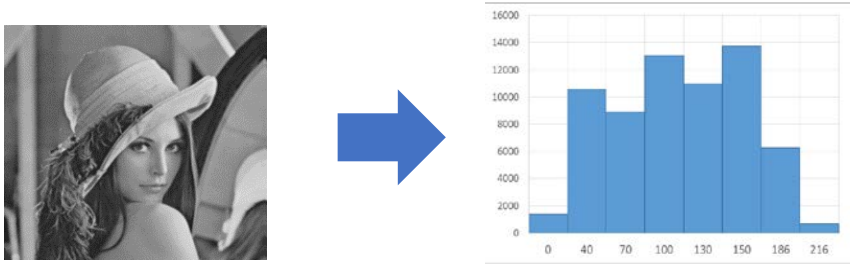
Configuration data file			r_drp_histogram.dat	
Supported version			0.90	
Configuration data size (byte)			82496	
Header file			r_drp_histogram.h	
Parameter	Structure name			
	r_drp_histogram_t			
	Member name	Type	Description	
	src	uint32_t	Input data address	
	dst	uint32_t	Output data address	
	data_size	uint32_t	Amount of input data (bytes)	
	mask	uint32_t	Masked data address	
	ranges	uint32_t	Address of the area holding the bin-width specification for the histogram	
	hist_size	uint16_t	Number of bins for the histogram	
	accumulate	uint8_t	Accumulation flag (0: initialization, 1: accumulation)	
I/O details	Input data	Address:	Specified by src. (Specify an address that differs from dst, mask, or ranges)	
		Amount of data:	Specified by data_size. (256 to 1,228,800)	
		Format:	8 bits (1 byte per datum)	
		Data size:	data_size x 1 byte	
	Output data	Address:	Specified by dst. (Specify an address that differs from src, mask, or ranges)	
		Number of bins:	Specified by hist_size. (1 to 256)	
		Format:	Frequency (represented by 4 bytes per bin) When the setting of the accumulation flag "accumulate" is for accumulation, the existing values for frequency are read out and set as the initial values of each of the bins in the region specified by dst. If a value exceeds the maximum value that can be represented by uint32_t, the value is limited to this maximum value. Refer to the description for details.	
		Data size:	hist_size x 4 bytes	

	Bin specification	Address:	Specified by ranges. (Specify an address that differs from src, dst, or mask.)
		Number of the bin area:	hist_size + 1
		Format:	16 bits (0 to 256) Set the lower limit for the 0th bin to the address specified by ranges +0 (bytes). Set the upper limit for the 0th bin to the address specified by ranges +2 (bytes). This function sets the lower limit for the 1st bin to the value of the address specified by ranges +2 (bytes).
		Data size:	(hist_size + 1) x 2 bytes
		Description	Set the upper and lower limits for all bins. For the i-th bin, the value becomes the address specified by ranges + i x 2 (bytes) or more, and less than the address specified by ranges + i x 2 + 2 (bytes). Specify the number of values specified by ranges to hist_size + 1. Refer to the description for details.
	Masked data	Address:	Specified by mask. (Specify an address that differs from src, dst, or ranges) If 0 is specified to mask, the mask function is disabled.
		Amount of data:	Same as input data.
		Format:	8 bits (1 byte per datum) Only when a value other than 0 is specified, the histogram is counted.
		Data size:	Same as input data.
		Description	The input data to which other than 0 is specified are counted for the histogram. Refer to the description for details.
Number of tiles	2		
Segmented processing	Not supported However, segmented processing can be set up in combination with processing by the CPU. Refer to the description for details.		

Description

This function calculates a histogram from the image data at the address specified by src and outputs the result to the address specified by dst. Specifying the data size (= width x height) of the image as data_size enables the input of an image as follows.

The bin areas for this function are specified by using hist_size and ranges.



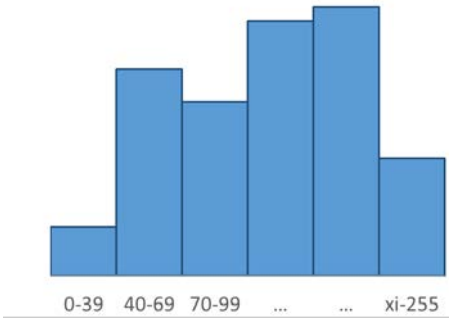
Input data Output data

To set the upper and lower limits for the hist_size bins, specify the bin areas for the (hist_size + 1) bins.

The lower limit for the i-th bin becomes range + i x 2.

The upper limit for the i-th bin becomes range + (i + 1) x 2.

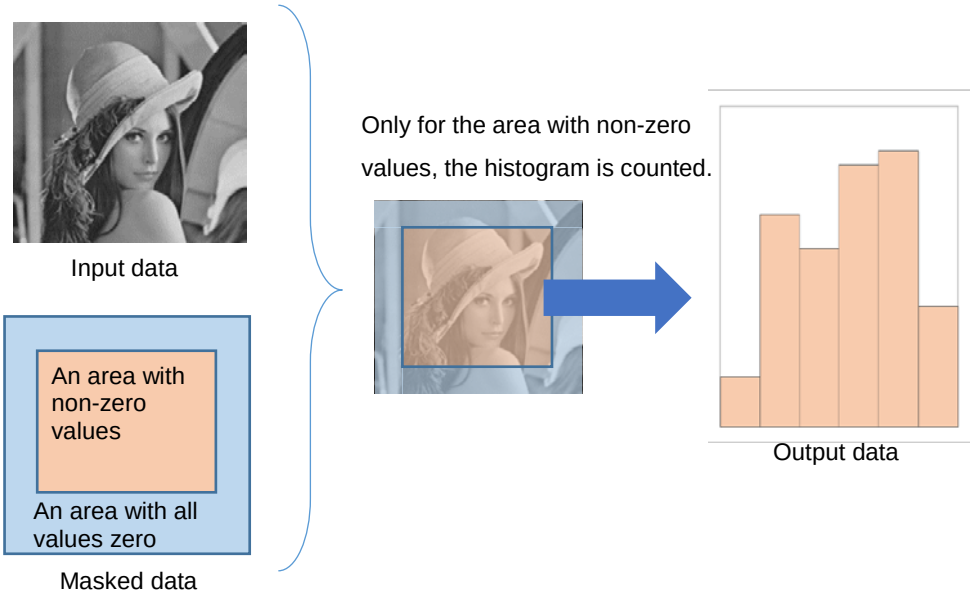
An example of specifying (i + 1) bins is shown below. In the example, for the i-th bin, $_i$ is set as the lower limit, and 255 is specified as the upper limit.



Address	Bin specification	
ranges+0	0	0th bin
ranges+2	40	1st bin
ranges+4	70	2nd bin
ranges+6	100	
:	:	
ranges+i×2	x_i	i-th bin
ranges+(i+1)×2	256	

This function enables masking of the values for counting to obtain the histogram by using mask.

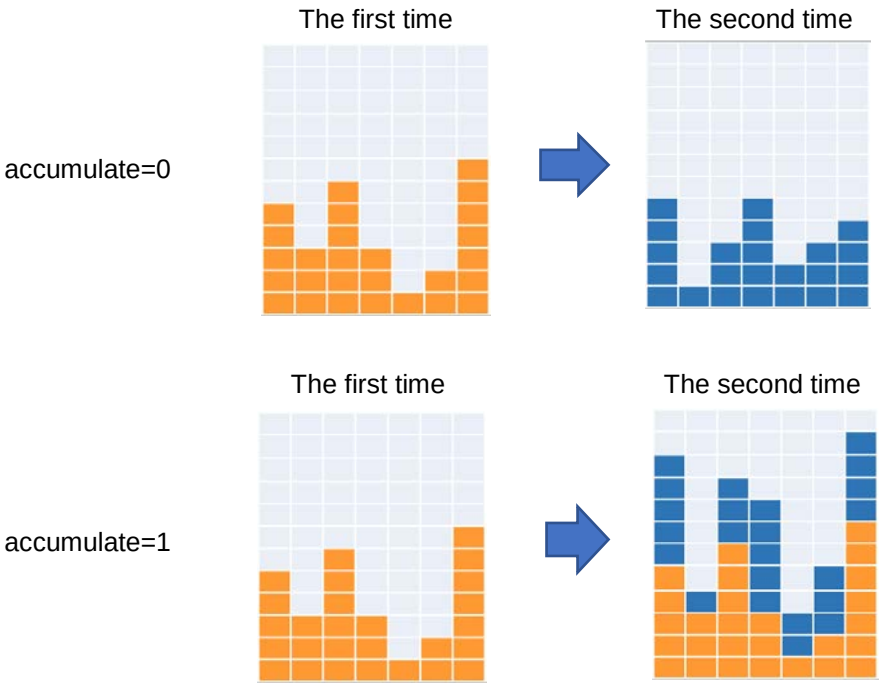
Pixels in areas for which 0 is specified the value are not counted in the histogram, and only values from areas having values other than 0 are counted in the histogram.



This function enables selection of the initial value or accumulated values of the histogram by using the variable accumulate.

Specifying accumulate as 1 causes reading of the existing results for a histogram at the address specified by dst, and the values thus obtained are set as the initial values. Specifying accumulate as 0 causes all of the initial values of the histogram to be set to 0.

Therefore, if accumulation is to be performed, the bin specifications (hist_size and ranges) cannot be changed from histogram to histogram. If the frequency exceeds 4,294,967,295 (= 2^{32} -1), the value is limited to 4,294,967,295.

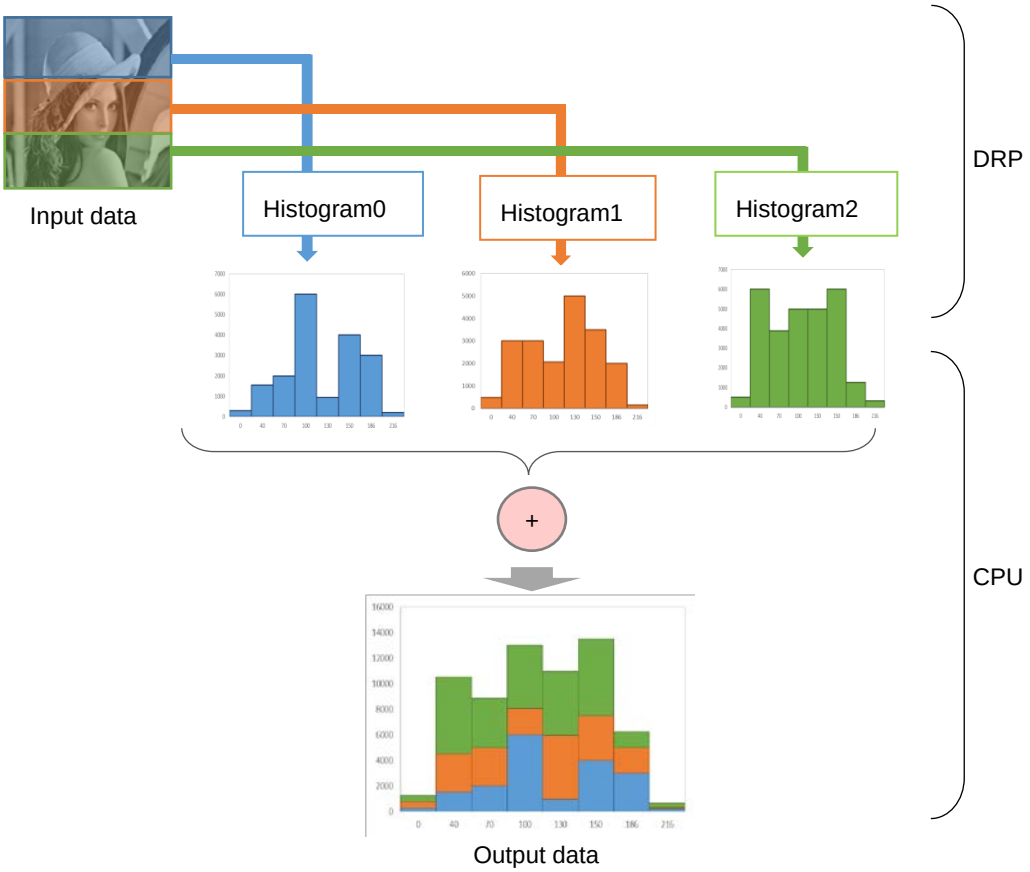


This function allows segmented processing with the aid of the CPU.

An example of three parallel flows of processing with the setting accumulate=0 is shown below.

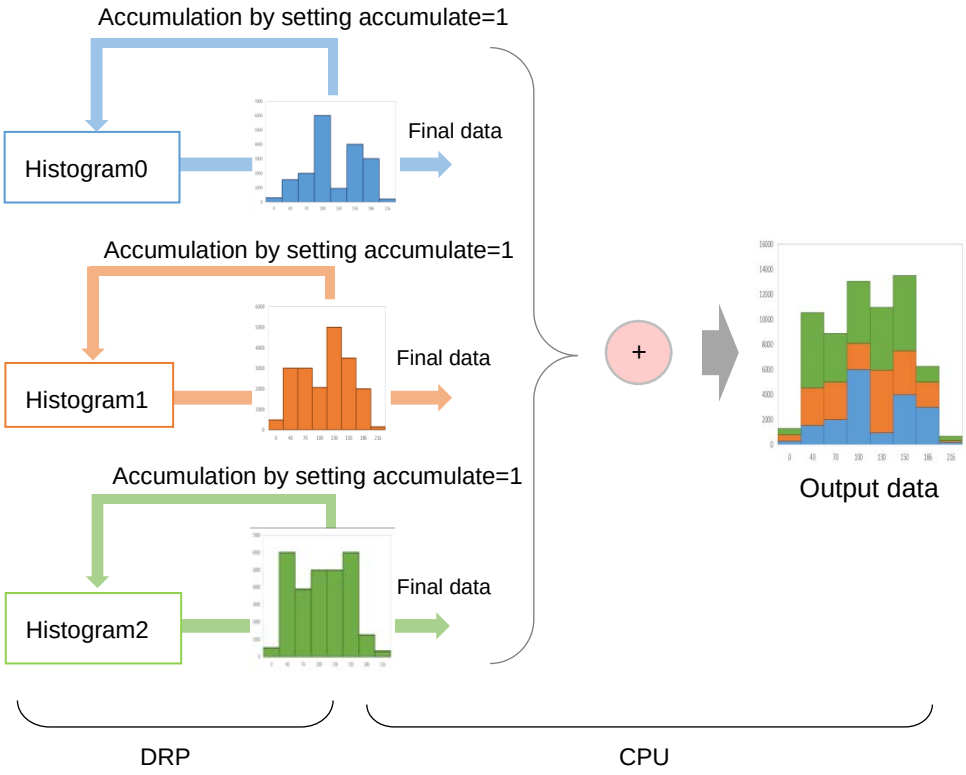
The input data are segmented into three areas: Histogram0, Histogram1, and Histogram2. The prescribed dsrc, dst, and mask, (and data_size as required) are specified for the respective areas. The parameters ranges and hist_size are to be the same.

The segmented processing is enabled by the CPU obtaining the total of the frequencies in corresponding bins in the dst areas for Histogram0, Histogram1, and Histogram2 after the DRP has calculated the histograms.



An example of three parallel flows of processing with the setting accumulate=1 is shown below.

If 1 is set for accumulate, segmented processing is enabled by adding up the frequencies of each bin in the dst area by CPU after the completion of accumulation in response to this setting of accumulate.



The processing performed by this function is equivalent to that of the OpenCV cv::calcHist function with specifying 1 to narrays argument, {0} to channels, 1 to dims, and false to uniform.

Reference URL: https://opencv.org/

Note	None

5. Using the DRP Library

To use this library, it is necessary to initialize the DRP, load configuration data, etc. Also, since the parameters are different for each configuration data, set the parameters based on the specification of the configuration data to be used. For application example of DRP library, refer to "RZ/A2M Group 2D Barcode Application Note (R01AN4503)".

6. Reference Documents

User's Manual: Hardware

RZ/A2M Group User's Manual: Hardware (R01UH0746)

(Download the latest version of the update or news from the Renesas Electronics website.)

User's Manual: Software

RZ/A2M Group DRP Driver User's Manual (R01US0355)

(Download the latest version of the update or news from the Renesas Electronics website.)

RZ/A2M Group 2D Barcode Sample Program Application Note (R01AN4503)

(Download the latest version of the update or news from the Renesas Electronics website.)

User's Manual: Development environment

For the Renesas Electronics integrated development environment (e^2 studio), visit the Renesas Electronics website to download the latest version.

Technical Update/Technical News

(Download the latest version of the update or news from the Renesas Electronics website.)

Revision History	RZ/A2M Group DRP Library User's Manual

Rev.	Date	Description	
		Page	Summary
1.00	Sep. 28, 2018	—	First Edition issued
1.01	Dec. 28, 2018	7	Following functions were added to Table 1.1 DRP Library Functions. (1) Prewitt (2) Opening (3) Closing (4) ResizeBilinearFixed (5) ResizeNearest (6) CircleFitting (7) Histogram
		9	2 Operation Conditions, The version of RENESAS e^2 studio was changed to 7.3.0.
		10, 11	3 File Structure, The configuration data and header files were added.
		12	4.1 How to Read the DRP Library Reference, An explanation for segmented processing was added.
		13	4.2 Simple ISP, section was added.
		20	4.3.1BinarizationFixed, The reference URL in the description column was changed.
		27	4.3.4Dilate The explanations for the top and bottom in parameter column were changed. The reference URL in the description section was changed, and an explanation was added.
		29	4.3.5 Erode The explanations for the top and bottom in parameter column were changed. The reference URL in the description section was changed, and an explanation was added.
		33	4.3.7 GaussianBlur The explanations for the top and bottom in parameter column were changed. The reference URL in the description section was changed, and the explanation was changed.
		35	4.3.8 MedianBlur The explanations for the top and bottom in parameter column were changed. The reference URL in the description section was changed, and the explanation was changed.
		37	4.3.9 Sobel The explanations for the top and bottom in parameter column were changed. The explanations in the description column were changed.
		39	4.3.10 Prewitt, section was added.
		43	4.3.11 UnsharpMasking The explanations for the top and bottom in parameter column were changed. The reference URL in the description section was changed, and the explanation was changed.
		45	4.3.13 Opening section was added.
		48	4.3.14 Closing section was added.

Rev.	Date	Description	
		Page	Summary
1.01	Dec. 28, 2018	52	4.4.2 Bayer2Grayscale The explanations for the top and bottom in parameter column were changed. The reference URL in the description section was changed, and the explanation was changed.
		66	4.4.6 ResizeBilinearFixed The title was changed from ResizeBilinear to ResizeBilinearFixed. The descriptions of I/O details, Input image width, and Data size were corrected. The reference URL in the description section was changed.
		68	4.4.7 ResizeBilinear section was added.
		70	4.4.8 ResizeNearest section was added.
		75	4.5.1 CannyCalculate The explanations for the top and bottom in parameter column were changed. The reference URL in the description section was changed.
		79	4.5.3 CornerHarris The figure, reference URL, and explanation in the description column were changed.
		81	4.5.4 CircleFitting section was added.
		108	4.6.2 Histogram section was added.
1.02	Apr. 15, 2019	7	The following functions were added to Table 1.1 DRP Library Functions. ・ Laplacian ・ Bayer2Rgb ・ ImageRotate ・ Affine ・ MinutiaeExtract ・ MinutiaeDelete ・ Thinning ・ ReedSolomonGf8
		10	Configuration data and header files were added to 3. File Structure.
		41	4.3.11 Laplacian added.
		55	4.4.3 Bayer2Rgb added.
		63	4.4.5 ImageRotate added.
		72	4.4.9 Affine added.
		85	4.5.5 MinutiaeExtract added.
		92	4.5.6 MinutiaeDelete added.
		102	4.5.8 Thinning added.
		106	4.6.1 ReedSolomon ・ The maximum value of input data size src_size was changed. ・ Changes were made to I/O details, Description, and Note.
		108	4.6.2 ReedSolomonGf8 added.
1.03	May 31, 2019	46	4.3.13 HistogramNormalization added.
		49	4.3.14 HistogramNormalizationRgb added.
		69	4.4.4 Bayer2RgbColorCorrection added.
		74	4.4.6 CroppingRgb added.
		80	4.4.9 ResizeBilinearFixedRgb
		98	4.5.5 FindContours added

RZ/A2M Group DRP Library User’s Manual

Publication Date: Rev.1.00 Sep. 28, 2018
Rev.1.03 May. 31, 2019

Published by: Renesas Electronics Corporation

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709 Quantum Plaza, No.27 ZhichunLu, Haidian District, Beijing, 100191 P. R. China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, 200333 P. R. China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 5.0

RZ/A2M Group

Renesas Electronics Corporation

R01US0367EJ0103